



System Verilog Reference Card Version 1.0

[] Optional
| Alternative
{ } Grouping
¹ [] are necessary

1. Module

```
module MODID[{{PORTID,}}];  
    [input | output | inout [range]  
{PORTID,};]  
    [{declaration}]  
    [{parallel_statement}]  
    [specify_block]  
    [program_block ]  
endmodule  
module module_id (PORTS);  
    [input | output | inout [range]  
{PORTID,};]  
    module module_id (PORTS);  
    .....  
    module module_id (PORTS);  
endmodule
```

Module Instance

```
    module_name identifier  
(.port_name(signal), ...  
    ...,  
.signal | variable);  
    module_name identifier (.*);
```

2. Declaration

```
parameter {PARID = constexpr,};  
int [signed | unsigned] variable;  
real [signed | unsigned] variable;  
bit [size][size] variable_name;  
byte [size] variable name;  
string string_identifier;  
localparam data_type variable =  
(initialization | expr);  
const data_type variable =  
(initialization | expr);  
struct | union  
    { data_type declarations;;  
    } structure_identifier;  
  
task [ static | automatic  
]task_identifier ([ port_list ]);  
    [ input | output | inout | ref  
argument_names ];  
    task body;  
endtask [ : task_identifier ]  
  
function data_type | void [ [size]  
] function_id ([ port_list ]);  
    [ input | output | inout | ref  
argement_names ];  
    function body;
```

```
endfunction [: function_id]
```

3. Data Types

Integer Types
sortint - 2 state - 16 bit signed
int - 2 state - 32 bit signed
longint - 2 state - 64 bit signed
byte - 2 state - 8 bit signed
integer
bit - 2 state - user defined
vector size
logic - 4 state - user defined
vector sized
reg - 4 state - user defined
vector sized
integer - 4 state - 32 bit signed
time - 4 state - 64 bit unsigned
string variable_name [= initial value];
typedef data_type
userdefined_data_type;
enum [data_type]
{enum_name_declaration} variable;
typedef struct [packed | unpacked]
[signed | unsigned]
 { data_type_declaration; }
structure_identifier;
typedef union [**tagged**]
 { structure_declaration;
 data_type_declaration;
 union_declaration;
 } union_identifier;

4. Methods

4.1 String Methods

```
string_id.len ()  
string_id.putc (int index,string | byte
```



```
variable);
string_id.getc (int index);
string_id.toupper ();
string_id.tolower ();
string_id.compare ();
string_id.icompare ();
string_id.substr (int i1,int i2);
string_id.[atoi () | atohex () | atoct
() | atobin () | atoreal() ];
string_id.[itoa | hextoa | octtoa |
bintoa] (integer v);
string_id.realtoa (real variable);
```

4.2 Array Methods
array_id. size() | delete() ;

4.3 Associative Array Methods
array_id.num;
array_id.delete [(index)];
array_id.exists [(index)];
array_id.first [(string variable)];
array_id.last [(string variable)];
array_id.next [(string variable)];
array_id.prev [(string variable)];

4.4 Queue Methods
queue_id.size();
queue_id.insert (int index,
queue_type item);
queue_id.delete (int index);
queue_id.pop_front () ;
queue_id.pop_back () ;
queue_id.push_front () ;
queue_id.push_back () ;

4.5 Array Locator Methods
int variable = arr_id find | find_next
|

find_last_index with
(condition);
string variable = arr_id find_first |
find_last |
unique with
(condition)

4.6 Array Ordering Methods
array_id. sort | rsort | reverse |
shuffle;

4.7 Array Reduction Methods
array_id.sum | product | and | or |
xor;

4.8 Sampled value functions
\$sampled(expression [,
clocking_event])
\$rose(expression [, clocking_event])
\$fell(expression [, clocking_event])
\$stable(expression [, clocking_event])
\$past(expression1 [, number_of_ticks
[, expression2] [,
clocking_event])

4.9 System functions
\$onehot (<expression>)
\$onehot0(<expression>)
\$isunknown (<expression>)

5. Arrays
data_type [size] array_id [size]
[=initialization];

```
array_id = new [ [size] ] (array_id);
data_type array_id [index type];
data_type queue_id [$];
```

6. Operators

6.1 Assignment Operators
= | += | -= | *= | /= | %= |
&= | |= | ^= | <<= | >>= |
<<<= | >>>=

6.2 Conditional Operators
target_variable = (condition) ?
(expression | variable) :
(expression | variable);

6.3 Unary Operators
+ | - | ! | ~ | & | ~& | | | ~|
| ^ | ~^ | ^~ | ++ | --

6.4 Binary Operators
+ | - | * | / | % | == | != |
=== | !== | ?= | !?= | && |
|| | ** | < | <= | > | >= |
& | | | ^ | ^~ | ~^ | >> |
<< | >>> | <<<

7. Procedural Statements

```
initial [: block_label]
begin
    event control statements;
    procedural statements;
end [: block_label]
fork [:block_identifier]
statements;
join | join_none | join_any
always @ (sensitivity list)
```

```

begin
  procedural statements;
end
always_comb | always_latch |
always_ff
begin
  procedural statements;
end
do
  procedural statements;
while (condition);
for([data_type] variable=initialization
; condition ;
      [ ++ | -- ] variable [ ++
| -- ])
begin
  statements ;
end
foreach (array_id [index])
begin
  statements ;
end

```

8. CLASS

```

[ virtual ] class class_identifier [
parameter_port_list
  [ extends class_type [ (
list_of_arguments ) ] ];
  class_variable_declaration;
  [ extern ] class method_declaration;
  class_constraints;
endclass [ : class_identifier ]

function new [ port_list ];
[ super.new [ list_of_arguments ] ];
[ this.srandom( [ seed ] ) ]

```

```

[ functional_statements_or_null ] ;
endfunction [ : new ]

```

9. Random Constraints

9.1 Random variables:

```
rand | randc data_declaration
```

9.2 Constraint_declaration :

```

[ static ] constraint
constraint_identifier
{
  constraint_expression }

constraint_expression ::=
Distribution : expression dist {
dist_list } ;
Implication : expression ->
constraint_set
if...else : if ( expression )
constraint_set [ else constraint_set ]
Iterative : foreach ( array_identifier
[ loop_variables ] )
  set membership : identifier inside
{specify_range};
variable ordering: solve
identifier_list before identifier_list;

class_variable_identifier . randomize [
( [ variable_identifier_list
| null ] ) ] [
with constraint_block ]
object [ .class_random_variable
].rand_mode( bit on_off );
object [ .class_random_varaibel

```

```

].constraint_mode ( bit on_off );
std::randomize (
local_variable_identifier_list ) [ with
{constraint_block}];

```

Random number system function and methods:

```

data_variable = $urandom [ (int seed)
]
data_variable = $urandom_range( int
unsigned maxval, int
unsigned minval)
srandom ( int seed)

```

```

randcase randcase_item {
randcase_item } endcase
randsequence ( [ production_
identifier ] )
production
endsequence

```

10. Event

```

mailbox [# data_type |
user_defined_data_type ] mailbox_id;
event event_identifier;

-> hierarchical_event_identifier;
->> hierarchical_event_identifier;
@ (hierarchical_event_identifier);
wait
(hierarchical_event_identifier.triggered) ;
wait_order ( hierarchical_identifier [ ,
hierarchical_identifier ] )
[ else action_block ] ;

```

11. Clocking block

```
[ default ] clocking clocking_identifier
@(hierarchical_identifier);
    default input clocking_skew |
output clocking_skew | inout
```

```
clocking_skew;
    clocking_direction
list_of_variables;
    endclocking [ : clocking_identifier ]
```

Cycle delay

```
## integral_number | ## identifier |
## ( expression )
```

12. Program Block

```
program [ static | automatic ]
program_identifier
```

```
[ parameter_port_list ];
    endprogram [ : program_identifier ]
```

13. Assertions

Immediate assertions :

```
assert ( expression ) action_block
else statements
```

Concurrent assertions:

```
sequence sequence_id [ (
list_of_formals ) ];
[ @ hierarchical_id ]
sequence_expression;
endsequence [ : sequence_id ]
```

Sequence_expression :

```
Repetition in sequence :  
Consecutive repetition :
```

```
id_1 ##a id_2 [ * [min_value : ]
max_value ]1 ##b ....
```

Goto repetition :

```
Id_1 ##a id_2 [ -> min_value :
max_value ]1 ##b ...
```

Non-consecutive repetition :

```
Id_1 ##a id_2 [ = min_value :
max_value ]1 ##b....
```

and/intersect/or operation :

```
(Sequence | expression) and |
intersect | or (sequence
| expression)
```

first_match :

```
first_match ( sequence_expression
);
```

throughout :

```
(expression) throughout
(sequence);
```

detecting endpoints :

```
id_1 ##a sequence_id.ended
##b....
```

```
property property_id [ (
list_of_formals ) ]
[ clocking_event ] property_expr;
endproperty [ : property_id ]
```

property_expr :

```
implication : sequence_expr ->
property_expr |
```

```
sequence_expr ==>
property_expr
```

not/and/or operation :

```
not ( property_expr ) and | or
(property_expr)
```

```
assert property ( property_id )
action_block else statement;
assume property ( property_spec );
cover property ( property_spec )
statement_or_null;
```

14. Packages

```
package package_identifier;
package_body;
endpackage [ : package_identifier ]
```

15. Interfaces

```
interface interface_id [#
constant_declaration ] [ ( port_list ) ]
interface_variable_declaration;
[ clocking_block_declaration ]
[ modport_declaration ]
[ task declaration ]
[function declaration ]
endinterface
```

modport_declaration :

```
modport modport_id ( [ port_list ] [
,| clocking clocking_block_id ]
[,|
export/import task_id/function_id ] );
```

virtual interface declaration :

```
virtual interface_id
list_of_virtual_interface_decl;
```

16. COVERAGE

```
covergroup covergroup_identifier
[[{tf_port_list}]] [clocking_event |
```

```

@@(block_event_expression)] ;
  { {attribute_instance} cover_point |
cover_cross
  | {attribute_instance}
option.member_id = expression |

type_option.member_id = expression; }
endgroup [:covergroup_identifier]

cover_point:
coverpoint id_1 iff ( expression )
coverpoint id_2 {
  bins a = { specify_range } ;
  bins b = { [value : $] };
  wildcard bins c = {4'b10??};
  bins d[] = default;
  ignore_bins ignore_val = {range};
  illegal_bins illegal_val = {range};
}
coverpoint id_3 {
  bins a =
(transition_seq),(transition_seq)...;
  bins b = default sequence;
}

cover_cross:
cross cover_point_id_1,
cover_point_id_2...;
cross cover_point_id_1, cover_point_id_2
{
  bins a = binsof ( cover_point_id_1)
intersect { range };
  bins b = binsof (cover_point_id_1)&&
| || binsof
(cover_point_id_2);

```

```

ignore_bins c = binsof
(cover_point_id_1) && intersect{range}
| binsof (
cover_point_id_2 );
}

```

17. PARAMETERS

```

localparam data_type_or_implicit
list_of_param_assignments ;
parameter data_type_or_implicit
list_of_param_assignments
| parameter type
list_of_type_assignments
specparam [ packed_dimension ]
list_of_specparam_assignments ;

```

18. SYSTEM TASKS AND FUNCTION

```

$typeof (expression) | $typeof
(data_type)
$typename (expression) | $typename
(data_type)
$bits (expression) | $bits
(type_identifier)
$isunbounded (constant_expression)
$left | $right | $low | $high |
$increment | $size
(array_identifier,dimension_expression) |

$left | $right | $low | $high |
$increment | $size (type_identifier
[,dimension_expression]) | $dimensions
(array_identifier) |

$dimensions (type_identifier)

```

```

$fatal [ ( 0|1|2 [, string | expression {,
string | expression }])] ;
$error | $warning | $info [[([string |
expression {,string | expression}]])] ;

```

```

$asserton | $assertoff | $assertkill
[(levels
[,list_of_modules_or_assertions]]);

```

```

$writememb (" file_name "
,memory_name [,start_addr
[,finish_addr]]); |
$writememh (" file_name "
,memory_name [,start_addr
[,finish_addr]]);

```

19. DPI

```

import "DPI" [ context | pure ]
[c_identifier =] function_prototype;
| import "DPI" [ context ] [c_identifier
=] task_prototype;
| export "DPI" [c_identifier =] function
function_identifier ;
| export "DPI" [c_identifier =] task
task_identifier ;
function function_data_type
function_identifier ([tf_port_list])
task task_identifier ([tf_port_list])

```

eInfochips Limited.
11/A-B, Chandra Colony,
Off C.G. Road, Ellisbridge,
Ahmedabad 380 006
Tel: +91-79-2656 3705
Fax: +91-79-2656 0722

eInfochips, Inc.
4655 Old Ironsides Drive,
Suite 385,
Santa Clara, CA 95054
Tel: +1-408-496-1882
Fax: +1-801-650-1480

www.einfochips.com
info@einfochips.com

