Annex A

Formal syntax definition

(normative)

The formal syntax of Verilog HDL is described using Backus-Naur Form (BNF).

A.1 Source text

A.1.1 Library source text

library_text ::= { library_descriptions }
library_descriptions ::=
    library_declaration
  | include_statement
  | config_declaration
library_declaration ::=
    library library_identifier file_path_spec [{ **,** file_path_spec }]
    [**-incdir** file_path_spec [{ **,** file_path_spec }] **;**
file_path_spec ::= file_path
include_statement ::= **include** <file_path_spec> **;**

A.1.2 Configuration source text

config_declaration ::=
    config config_identifier **;**
    design_statement
    {config_rule_statement}
    endconfig
design_statement ::= **design** { [library_identifier.]cell_identifier } **;**
config_rule_statement ::=
    default_clause liblist_clause
  | inst_clause liblist_clause
  | inst_clause use_clause
  | cell_clause liblist_clause
  | cell_clause use_clause
default_clause ::= **default**
inst_clause ::= **instance** inst_name
inst_name ::= topmodule_identifier{.instance_identifier}
cell_clause ::= **cell** [library_identifier.]cell_identifier
liblist_clause ::= **liblist** [{library_identifier}]
use_clause ::= **use** [library_identifier.]cell_identifier[**:config**]

Copyright 2000 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

A.1.3 Module and primitive source text

```
source_text ::= { description }
description ::=
        module_declaration
      | udp_declaration
module_declaration ::=
      module_keyword module_identifier [ module_parameter_port_list ]
            [ list_of_ports ] [ attribute_instance ] ; { module_item }
            endmodule
module_keyword ::= module | macromodule
```

A.1.4 Module parameters and ports

```
module_parameter_port_list ::= # ( parameter_declaration { , parameter_declaration } )
list_of_ports ::=
        ( port { , port } )
      | ( port_definition { , port_definition } )
      | ( )
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 [ range_expression ]
port_definition ::=
        input_declaration
      | output_declaration
      | inout_declaration
```

A.1.5 Module items

```
module_item ::=
        module_item_declaration
      | module_or_generate_item
      | generated_instantiation
      | specify_block
module_item_declaration ::=
        parameter_declaration
      | local_parameter_declaration
      | specparam_declaration
      | input_declaration
      | output_declaration
      | inout_declaration
```


Copyright 2000 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

```
module_or_generate_item ::=
      module_or_generate_item_declaration
    | parameter_override
    | continuous_assign
    | gate_instantiation
    | udp_instantiation
    | module_instantiation
    | initial_construct
    | always_construct
    | attribute_instance
module_or_generate_item_declaration ::=
      net_declaration
    | reg_declaration
    | integer_declaration
    | real_declaration
    | time_declaration
    | realtime_declaration
    | event_declaration
    | genvar_declaration
    | task_declaration
    | function_declaration
parameter_override ::= defparam list_of_param_assignments ;
```

A.2 Declarations

A.2.1 Declaration types

A.2.1.1 Parameter declarations

```
local_parameter_declaration ::=
      localparam [ signed ] [ range ] list_of_param_assignments ;
    | localparam integer list_of_param_assignments ;
    | localparam real list_of_param_assignments ;
    | localparam realtime list_of_param_assignments ;
    | localparam time list_of_param_assignments ;
parameter_declaration ::=
      parameter [ signed ] [ range ] list_of_param_assignments ;
    | parameter integer list_of_param_assignments ;
    | parameter real list_of_param_assignments ;
    | parameter realtime list_of_param_assignments ;
    | parameter time list_of_param_assignments ;
specparam_declaration ::= specparam [ range ] list_of_specparam_assignments ;
```

A.2.1.2 Port declarations

```
inout_declaration ::= inout [ net_type ] [ signed ] [ range ]
          [ attribute_instance ] list_of_port_identifiers ;
input_declaration ::= input [ net_type ] [ signed ] [ range ]
          [ attribute_instance ] list_of_port_identifiers ;
```


Copyright 2000 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

output_declaration ::=
    output [net_type] [**signed**] [range]
        [attribute_instance] list_of_port_identifiers **;**
  | **output** [**reg**] [**signed**] [range]
        [attribute_instance] list_of_port_identifiers **;**
  | **output reg** [**signed**] [range]
        [attribute_instance] list_of_reg_port_identifiers **;**
  | **output** [output_reg_type]
        [attribute_instance] list_of_port_identifiers **;**
  | **output** output_reg_type
        [attribute_instance] list_of_reg_port_identifiers **;**

A.2.1.3 Type declarations

event_declaration ::= **event** list_of_event_identifiers **;**
genvar_declaration ::= **genvar** list_of_genvar_identifiers **;**
integer_declaration ::= **integer** [attribute_instance] list_of_register_identifiers **;**
net_declaration ::=
    net_type [**signed**]
        [delay3] [attribute_instance] list_of_net_identifiers **;**
  | net_type [drive_strength] [**signed**]
        [delay3] [attribute_instance] list_of_net_decl_assignments **;**
  | net_type [**vectored** | **scalared**] [**signed**]
        [range] [delay3] [attribute_instance] list_of_net_identifiers **;**
  | net_type [drive_strength] [**vectored** | **scalared**] [**signed**]
        [range] [delay3] [attribute_instance] list_of_net_decl_assignments **;**
  | **trireg** [charge_strength] [**signed**]
        [delay3] [attribute_instance] list_of_net_identifiers **;**
  | **trireg** [drive_strength] [**signed**]
        [delay3] [attribute_instance] list_of_net_decl_assignments **;**
  | **trireg** [charge_strength] [**vectored** | **scalared**] [**signed**]
        [range] [delay3] [attribute_instance] list_of_net_identifiers **;**
  | **trireg** [drive_strength] [**vectored** | **scalared**] [**signed**]
        [range] [delay3] [attribute_instance] list_of_net_decl_assignments **;**
real_declaration ::= **real** list_of_real_identifiers **;**
realtime_declaration ::= **realtime** list_of_real_identifiers **;**
reg_declaration ::= **reg** [**signed**] [range]
        [attribute_instance] list_of_register_identifiers **;**
time_declaration ::= **time** [attribute_instance] list_of_register_identifiers **;**

A.2.2 Declaration data types

A.2.2.1 Net and register types

memory_type ::= memory_identifier dimension
net_type ::=
    supply0 | **supply1**
  | **tri** | **triand** | **trior** | **tri0** | **tri1**
  | **wire** | **wand** | **wor**
output_reg_type ::= **integer** | **real** | **realtime** | **time**

Copyright 2000 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

```
real_type ::=
        real_identifier [ = constant_expression ]
      | real_identifier [ dimension { dimension }]
register_type ::=
        register_identifier [ = constant_expression ]
      | register_identifier [ dimension { dimension }]
```

A.2.2.2 Strengths

```
drive_strength ::=
        ( strength0 , strength1 )
      | ( strength1 , strength0 )
      | ( strength0 , highz1 )
      | ( strength1 , highz0 )
      | ( highz0 , strength1 )
      | ( highz1 , strength0 )
strength0 ::= supply0 | strong0 | pull0 | weak0
strength1 ::= supply1 | strong1 | pull1 | weak1
charge_strength ::= ( small ) | ( medium ) | ( large )
```

A.2.2.3 Delays

```
delay3 ::= # delay_value | # ( delay_value [ , delay_value [ , delay_value ] ] )
delay2 ::= # delay_value | # ( delay_value [ , delay_value ] )
delay_value ::=
        unsigned_number
      | parameter_identifier
      | specparam_identifier
      | mintypmax_expression
```

A.2.3 Declaration lists

```
list_of_event_identifiers ::= event_identifier [ dimension { dimension }]
            { , event_identifier [ dimension { dimension }] }
list_of_genvar_identifiers ::= genvar_identifier { , genvar_identifier }
list_of_local_param_assignments ::= param_assignment { , param_assignment }
list_of_net_decl_assignments ::= net_decl_assignment { , net_decl_assignment }
list_of_net_identifiers ::= net_identifier [ dimension { dimension }]
            { , net_identifier [ dimension { dimension }] }
list_of_param_assignments ::= param_assignment [ range ] { , param_assignment }
list_of_port_identifiers ::= port_identifier { , port_identifier }
list_of_real_identifiers ::= real_type { , real_type }
list_of_register_identifiers ::=
        register_type { , register_type | memory_type }
      | memory_type { , register_type | memory_type }
list_of_reg_port_identifiers ::= port_identifier [ = constant_expression ]
                  { , port_identifier [ = constant_expression ] }
list_of_specparam_assignments ::= specparam_assignment { , specparam_assignment }
```


Copyright 2000 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

A.2.4 Declaration assignments

net_decl_assignment ::= net_identifier **=** expression
param_assignment ::= parameter_identifier **=** constant_expression
specparam_assignment ::=
    specparam_identifier **=** constant_mintypmax_expression
  | pulse_control_specparam
pulse_control_specparam ::=
    PATHPULSE$ = (reject_limit_value [**,** error_limit_value] **) ;**
  | **PATHPULSE$**specify_input_terminal_descriptor$specify_output_terminal_descriptor
        = (reject_limit_value [**,** error_limit_value] **) ;**
error_limit_value ::= limit_value
reject_limit_value ::= limit_value
limit_value ::= constant_mintypmax_expression

A.2.5 Declaration ranges

dimension ::= **[** dimension_constant_expression **:** dimension_constant_expression **]**
range ::= **[** msb_constant_expression **:** lsb_constant_expression **]**

A.2.6 Function declarations

function_declaration ::=
    function [**automatic**] [**signed**] [range_or_type]
      function_identifier [attribute_instance] **;**
    function_item_declaration { function_item_declaration }
    function_statement
    endfunction
  | **function** [**automatic**] [**signed**] [range_or_type]
      function_identifier **(** function_port_list **)** [attribute_instance] **;**
    block_item_declaration { block_item_declaration }
    function_statement
    endfunction
function_item_declaration ::=
    block_item_declaration
  | input_declaration
function_port_list ::= input_declaration { **,** input_declaration }
range_or_type ::= range | **integer** | **real** | **realtime** | **time**

A.2.7 Task declarations

task_declaration ::=
    task [**automatic**] task_identifier [attribute_instance] **;**
    { task_item_declaration }
    statement
    endtask

Copyright 2000 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

```
        | task [ automatic ] task_identifier ( task_port_list ) [ attribute_instance ] ;
          { block_item_declaration }
          statement
          endtask
task_item_declaration ::=
          block_item_declaration
        | input_declaration
        | output_declaration
        | inout_declaration
task_port_list ::= task_port_item { , task_port_item }
task_port_item ::=
          input_declaration
        | output_declaration
        | inout_declaration
```

A.2.8 Block item declarations

```
block_item_declaration ::=
          block_reg_declaration
        | event_declaration
        | integer_declaration
        | local_parameter_declaration
        | parameter_declaration
        | real_declaration
        | realtime_declaration
        | time_declaration
block_reg_declaration ::= reg [ signed ] [ range ]
                [ attribute_instance ] list_of_block_register_identifiers ;
list_of_block_register_identifiers ::=
          block_register_type { , block_register_type | memory_type }
        | memory_type { , block_register_type | memory_type }
block_register_type ::=
          register_identifier
        | register_identifier [ dimension { dimension }]
```

A.3 Primitive instances

A.3.1 Primitive instantiation and instances

```
gate_instantiation ::=
          cmos_switchtype [delay3] [ attribute_instance ]
                cmos_switch_instance { , cmos_switch_instance } ;
        | enable_gatetype [drive_strength] [delay3] [ attribute_instance ]
                enable_gate_instance { , enable_gate_instance } ;
        | mos_switchtype [delay3] [ attribute_instance ]
                mos_switch_instance { , mos_switch_instance } ;
        | n_input_gatetype [drive_strength] [delay2] [ attribute_instance ]
                n_input_gate_instance { , n_input_gate_instance } ;
```


Copyright 2000 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

| n_output_gatetype [drive_strength] [delay2] [attribute_instance]
n_output_gate_instance { , n_output_gate_instance } ;
| pass_en_switchtype [delay2] [attribute_instance]
pass_enable_switch_instance { , pass_enable_switch_instance } ;
| pass_switchtype [attribute_instance]
pass_switch_instance { , pass_switch_instance } ;
| **pulldown** [pulldown_strength] [attribute_instance]
pull_gate_instance { , pull_gate_instance } ;
| **pullup** [pullup_strength] [attribute_instance]
pull_gate_instance { , pull_gate_instance } ;

cmos_switch_instance ::= [name_of_gate_instance] (output_terminal , input_terminal ,
ncontrol_terminal , pcontrol_terminal)
enable_gate_instance ::= [name_of_gate_instance] (output_terminal , input_terminal , enable_terminal)
mos_switch_instance ::= [name_of_gate_instance] (output_terminal , input_terminal , enable_terminal)
n_input_gate_instance ::= [name_of_gate_instance] (output_terminal , input_terminal { , input_terminal })
n_output_gate_instance ::= [name_of_gate_instance] (output_terminal { , output_terminal } , input_terminal)
pass_switch_instance ::= [name_of_gate_instance] (inout_terminal , inout_terminal)
pass_enable_switch_instance ::= [name_of_gate_instance] (inout_terminal , inout_terminal , enable_terminal)
pull_gate_instance ::= [name_of_gate_instance] (output_terminal)
name_of_gate_instance ::= gate_instance_identifier [range]

A.3.2 Primitive strengths

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

A.3.3 Primitive terminals

enable_terminal ::= scalar_expression
inout_terminal ::= terminal_identifier | terminal_identifier [constant_expression]
input_terminal ::= scalar_expression
ncontrol_terminal ::= scalar_expression
output_terminal ::= terminal_identifier | terminal_identifier [constant_expression]
pcontrol_terminal ::= scalar_expression

A.3.4 Primitive gate and switch types

cmos_switchtype ::= **cmos** | **rcmos**
enable_gatetype ::= **bufif0** | **bufif1** | **notif0** | **notif1**
mos_switchtype ::= **nmos** | **pmos** | **rnmos** | **rpmos**
n_input_gatetype ::= **and** | **nand** | **or** | **nor** | **xor** | **xnor**
n_output_gatetype ::= **buf** | **not**

Copyright 2000 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

pass_en_switchtype ::= **tranif0** | **tranif1** | **rtranif1** | **rtranif0**
pass_switchtype ::= **tran** | **rtran**

A.4 Module and generated instantiation

A.4.1 Module instantiation

module_instantiation ::=
    module_identifier [parameter_value_assignment] [attribute_instance]
      module_instance { **,** module_instance } **;**
parameter_value_assignment ::= **# (** list_of_parameter_assignments **)**
list_of_parameter_assignments ::=
    ordered_parameter_assignment { **,** ordered_parameter_assignment } |
    named_parameter_assignment { **,** named_parameter_assignment }
ordered_parameter_assignment ::= expression
named_parameter_assignment ::= **.** parameter_identifier **(** [expression] **)**
module_instance ::= name_of_instance **(** [list_of_module_connections] **)**
name_of_instance ::= module_instance_identifier [range]
list_of_port_connections ::=
    ordered_port_connection { **,** ordered_port_connection }
  | named_port_connection { **,** named_port_connection }
ordered_port_connection ::= [expression] [attribute_instance]
named_port_connection ::= **.**port_identifier **(** [expression] **)** [attribute_instance]

A.4.2 Generated instantiation

generated_instantiation ::= **generate** { generate_item } **endgenerate**
generate_item_or_null ::= generate_item | **;**
generate_item ::=
    generate_conditional_statement
  | generate_case_statement
  | generate_loop_statement
  | generate_block
  | module_or_generate_item
generate_conditional_statement ::=
    if (genvar_expression **)** generate_item_or_null [**else** generate_item_or_null]
generate_case_statement ::= **case (** genvar_expression **)**
        genvar_case_item { genvar_case_item } **endcase**
genvar_case_item ::= genvar_expression { **,** genvar_expression } **:**
        generate_item_or_null | **default** [**:**] generate_item_or_null
generate_loop_statement ::= **for (** genvar_assignment **;** genvar_expression **;** genvar_assignment **)** generate_item
        begin : generate_block_identifier { generate_item } **end**
genvar_assignment ::= genvar_identifier **=** genvar_expression
generate_block ::= **begin** [**:** generate_block_identifier] { generate_item } **end**

Copyright 2000 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

A.5 UDP declaration and instantiation

A.5.1 UDP declaration

udp_declaration ::=
primitive udp_identifier **(** udp_port_list **)** [attribute_instance] **;**
udp_port_declaration { udp_port_declaration }
udp_body
endprimitive
| **primitive** udp_identifier **(** udp_declaration_port_list **)** [attribute_instance] **;**
udp_body
endprimitive

A.5.2 UDP ports

udp_port_list ::= output_port_identifier **,** input_port_identifier { **,** input_port_identifier }
udp_declaration_port_list ::=
udp_output_declaration **,** udp_input_declaration { **,** udp_input_declaration }
udp_port_declaration ::=
udp_output_declaration
| udp_input_declaration
| udp_reg_declaration
udp_output_declaration ::=
output [attribute_instance] port_identifier **;**
| **output** reg [attribute_instance] port_identifier [**=** constant_expression] **;**
udp_input_declaration ::=
input [attribute_instance] list_of_port_identifiers **;**
udp_reg_declaration ::= **reg** register_identifier **;**

A.5.3 UDP body

udp_body ::= combinational_body | sequential_body
combinational_body ::= **table** combinational_entry { combinational_entry } **endtable**
combinational_entry ::= level_input_list **:** output_symbol **;**
sequential_body ::= [udp_initial_statement] **table** sequential_entry { sequential_entry } **endtable**
udp_initial_statement ::= **initial** output_port_identifier **=** init_val **;**
init_val ::= **1'b0** | **1'b1** | **1'bx** | **1'bX** | **1'B0** | **1'B1** | **1'Bx** | **1'BX** | **1** | **0**
sequential_entry ::= seq_input_list **:** current_state **:** next_state **;**
seq_input_list ::= level_input_list | edge_input_list
level_input_list ::= level_symbol { level_symbol }
edge_input_list ::= { level_symbol } edge_indicator { level_symbol }
edge_indicator ::= **(** level_symbol level_symbol **)** | edge_symbol
current_state ::= level_symbol
next_state ::= output_symbol | **-**
output_symbol ::= **0** | **1** | **x** | **X**
level_symbol ::= **0** | **1** | **x** | **X** | **?** | **b** | **B**
edge_symbol ::= **r** | **R** | **f** | **F** | **p** | **P** | **n** | **N** | ***

Copyright 2000 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

A.5.4 UDP instantiation

```
udp_instantiation ::= udp_identifier [ drive_strength ] [ delay2 ]
                                    [attribute_instance] udp_instance { , udp_instance } ;
udp_instance ::= [ name_of_udp_instance ] ( output_terminal , input_terminal
                                    { , input_terminal } )
name_of_udp_instance ::= udp_instance_identifier [ range ]
```

A.6 Behavioral statements

A.6.1 Continuous assignment statements

```
continuous_assign ::= assign [ drive_strength ] [ delay3 ] list_of_net_assignments ;
list_of_net_assignments ::= net_assignment { , net_assignment }
net_assignment ::= net_lvalue = expression
```

A.6.2 Procedural blocks and assignments

```
initial_construct ::= initial [ attribute_instance ] statement
always_construct ::= always [ attribute_instance ] statement
blocking_assignment ::= reg_lvalue = [ delay_or_event_control ] expression
non_blocking_assignment ::= reg_lvalue <= [ delay_or_event_control ] expression
procedural_continuous_assignments ::=
      assign reg_assignment ;
    | deassign reg_lvalue ;
    | force reg_assignment ;
    | force net_assignment ;
    | release reg_lvalue ;
    | release net_lvalue ;
function_blocking_assignment ::= reg_lvalue = expression
function_statement_or_null ::= function_statement | ;
```

A.6.3 Parallel and sequential blocks

```
function_seq_block ::= begin [ : block_identifier ]
            [ { block_item_declaration } ] { function_statement } end
reg_assignment ::= reg_lvalue = expression
par_block ::= fork [ : block_identifier [ attribute_instance ]
              { block_item_declaration } ] { statement } join
seq_block ::= begin [ : block_identifier [ attribute_instance ]
              { block_item_declaration } ] { statement } end
```

A.6.4 Statements

```
statement ::=
      blocking_assignment ;
    | non_blocking_assignment ;
    | procedural_continuous_assignments ;
```


Copyright 2000 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

```
        | procedural_timing_control_statement
        | case_statement
        | loop_statement
        | wait_statement
        | disable_statement
        | event_trigger
        | seq_block
        | par_block
        | task_enable
        | system_task_enable
statement_or_null ::= statement | ;
function_statement ::=
         function_blocking_assignment ;
        | function_seq_block
        | function_conditional_statement
        | function_case_statement
        | function_loop_statement
        | disable_statement
        | system_task_enable
```

A.6.5 Timing control statements

```
delay_control ::=
         # delay_value
        | # ( mintypmax_expression )
delay_or_event_control ::=
         delay_control
        | event_control
        | repeat ( expression ) event_control
disable_statement ::=
         disable hierarchical_task_identifier ;
        | disable hierarchical_block_identifier ;
event_control ::=
         @ event_identifier
        | @ ( event_expression )
        | @*
        | @ (*)
event_trigger ::=
         -> hierarchical_event_identifier ;
event_expression ::=
         expression
        | hierarchical_identifier
        | posedge expression
        | negedge expression
        | event_expression or event_expression
        | event_expression , event_expression
wait_statement ::=
         wait ( expression ) statement_or_null
```


Copyright 2000 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

A.6.6 Conditional statements

conditional_statement ::=
if (expression **)** [attribute_instance]
statement_or_null [**else** statement_or_null]
| if_else_if_statement
if_else_if_statement ::=
if (expression **)** [attribute_instance] statement_or_null
{ **else if (** expression **)** [attribute_instance] statement_or_null }
[**else** statement_or_null]
function_conditional_statement ::=
if (expression **)** function_statement_or_null [**else** function_statement_or_null]
| function_if_else_if_statement
function_if_else_if_statement ::=
if (expression **)** function_statement_or_null
{ **else if (** expression **)** function_statement_or_null }
[**else** function_statement_or_null]

A.6.7 Case statements

case_statement ::=
case (expression **)** [attribute_instance]
case_item [case_item] **endcase**
| **casez (** expression **)** [attribute_instance]
case_item [case_item] **endcase**
| **casex (** expression **)** [attribute_instance]
case_item [case_item] **endcase**
case_item ::=
expression { **,** expression } **:** statement_or_null
| **default** [**:**] statement_or_null
function_case_statement ::=
case (expression **)** case_item [function_case_item] **endcase**
| **casez (** expression **)** case_item [function_case_item] **endcase**
| **casex (** expression **)** case_item [function_case_item] **endcase**
function_case_item ::=
expression { **,** expression } **:** function_statement_or_null
| **default** [**:**] function_statement_or_null

A.6.8 Looping statements

function_loop_statement ::=
forever function_statement
| **repeat (** expression **)** function_statement
| **while (** expression **)** function_statement
| **for (** reg_assignment **;** expression **;** reg_assignment **)** function_statement
loop_statement ::=
forever statement
| **repeat (** expression **)** statement

Copyright 2000 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

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

A.6.9 Task enable statements

system_task_enable ::= system_task_identifier [**(** expression { **,** expression } **)**] **;**
task_enable ::= hierarchical_task_identifier [attribute_instance]
[**(** expression { **,** expression } **)**] **;**

A.7 Specify section

A.7.1 Specify block declaration

specify_block ::= **specify** { specify_item } **endspecify**
specify_item ::=
specparam_declaration
| pulsestyle_declaration
| showcancelled_declaration
| path_declaration
| system_timing_check
pulsestyle_declaration ::=
pulsestyle_onevent list_of_path_outputs
| **pulsestyle_ondetect** list_of_path_outputs
showcancelled_declaration ::=
showcancelled list_of_path_outputs
| **noshowcancelled** list_of_path_outputs

A.7.2 Specify path declarations

path_declaration ::=
simple_path_declaration **;**
| edge_sensitive_path_declaration **;**
| state_dependent_path_declaration **;**
simple_path_declaration ::=
parallel_path_description **=** path_delay_value
| full_path_description **=** path_delay_value
parallel_path_description ::=
(specify_input_terminal_descriptor [polarity_operator] **=>** specify_output_terminal_descriptor **)**
full_path_description ::=
(list_of_path_inputs [polarity_operator] ***>** list_of_path_outputs **)**
list_of_path_inputs ::=
specify_input_terminal_descriptor { **,** specify_input_terminal_descriptor }
list_of_path_outputs ::=
specify_output_terminal_descriptor { **,** specify_output_terminal_descriptor }

Copyright 2000 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

A.7.3 Specify block terminals

```
specify_input_terminal_descriptor ::=
        input_identifier
      | input_identifier [ constant_expression ]
      | input_identifier [ range_expression ]
specify_output_terminal_descriptor ::=
        output_identifier
      | output_identifier [ constant_expression ]
      | output_identifier [ range_expression ]
input_identifier ::= input_port_identifier | inout_port_identifier
output_identifier ::= output_port_identifier | inout_port_identifier
```

A.7.4 Specify path delays

```
path_delay_value ::=
        list_of_path_delay_expressions
      | ( list_of_path_delay_expressions )
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
t_path_delay_expression ::= path_delay_expression
trise_path_delay_expression ::= path_delay_expression
tfall_path_delay_expression ::= path_delay_expression
tz_path_delay_expression ::= path_delay_expression
t01_path_delay_expression ::= path_delay_expression
t10_path_delay_expression ::= path_delay_expression
t0z_path_delay_expression ::= path_delay_expression
tz1_path_delay_expression ::= path_delay_expression
t1z_path_delay_expression ::= path_delay_expression
tz0_path_delay_expression ::= path_delay_expression
t0x_path_delay_expression ::= path_delay_expression
tx1_path_delay_expression ::= path_delay_expression
t1x_path_delay_expression ::= path_delay_expression
tx0_path_delay_expression ::= path_delay_expression
txz_path_delay_expression ::= path_delay_expression
tzx_path_delay_expression ::= path_delay_expression
path_delay_expression ::= constant_mintypmax_expression
edge_sensitive_path_declaration ::=
        parallel_edge_sensitive_path_description = path_delay_value
      | full_edge_sensitive_path_description = path_delay_value
```


Copyright 2000 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

parallel_edge_sensitive_path_description ::=
([edge_identifier] specify_input_terminal_descriptor =>
specify_output_terminal_descriptor [polarity_operator] : data_source_expression)
full_edge_sensitive_path_description ::=
([edge_identifier] list_of_path_inputs *>
list_of_path_inputs [polarity_operator] : data_source_expression)
data_source_expression ::= expression
edge_identifier ::= **posedge** | **negedge**
state_dependent_path_declaration ::=
if (conditional_expression **)** simple_path_declaration
| **if (** conditional_expression **)** edge_sensitive_path_declaration
| **ifnone** simple_path_declaration
polarity_operator ::= + | -

A.7.5 System timing checks

A.7.5.1 System timing check commands

system_timing_check ::=
$setup_timing_check
| $hold _timing_check
| $setuphold_timing_check
| $recovery_timing_check
| $removal_timing_check
| $recrem_timing_check
| $skew_timing_check
| $timeskew_timing_check
| $fullskew_timing_check
| $period_timing_check
| $width_timing_check
| $nochange_timing_check
$setup_timing_check ::=
$setup (data_event **,** reference_event **,** timing_check_limit [**,** [notify_register]] **) ;**
$hold _timing_check ::=
$hold (reference_event **,** data_event **,** timing_check_limit [**,** [notify_register]] **) ;**
$setuphold_timing_check ::=
$setuphold (reference_event **,** data_event **,** timing_check_limit **,** timing_check_limit
[**,** [notify_register] [**,** [stamptime_condition] [**,** [checktime_condition]
[**,** [delayed_reference] [**,** [delayed_data]]]]]] **) ;**
$recovery_timing_check ::=
$recovery (reference_event **,** data_event **,** timing_check_limit [**,** [notify_register]] **) ;**
$removal_timing_check ::=
$removal (reference_event **,** data_event **,** timing_check_limit [**,** [notify_register]] **) ;**
$recrem_timing_check ::=
$recrem (reference_event **,** data_event **,** timing_check_limit **,** timing_check_limit
[**,** [notify_register] [**,** [stamptime_condition] [**,** [checktime_condition]
[**,** [delayed_reference] [**,** [delayed_data]]]]]] **) ;**
$skew_timing_check ::=
$skew (reference_event **,** data_event **,** timing_check_limit [**,** [notify_register]] **) ;**

Copyright 2000 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

```
$timeskew_timing_check ::=
      $timeskew ( reference_event , data_event , timing_check_limit
                  [ , [ notify_register ] [ , [ event_based_flag ] [ , [ remain_active_flag ] ] ] ] ) ;
$fullskew_timing_check ::=
      $fullskew ( reference_event , data_event , timing_check_limit , timing_check_limit
                  [ , [ notify_register ] [ , [ event_based_flag ] [ , [ remain_active_flag ] ] ] ] ) ;
$period_timing_check ::=
      $period ( controlled_reference_event , timing_check_limit [ , [ notify_register ] ] ) ;
$width_timing_check ::=
      $width ( controlled_reference_event , timing_check_limit ,
                  threshold [ , [ notify_register ] ] ) ;
$nochange_timing_check ::=
      $nochange ( reference_event , data_event , start_edge_offset ,
                  end_edge_offset [ , [ notify_register ] ] ) ;
```

A.7.5.2 System timing check command arguments

```
checktime_condition ::= mintypmax_expression
controlled_reference_event ::= controlled_timing_check_event
data_event ::= timing_check_event
delayed_data ::=
      terminal_identifier
    | terminal_identifier [ constant_mintypmax_expression ]
delayed_reference ::=
      terminal_identifier
    | terminal_identifier [ constant_mintypmax_expression ]
end_edge_offset ::= mintypmax_expression
event_based_flag ::= constant_expression
notify_register ::= register_identifier
reference_event ::= timing_check_event
remain_active_flag ::= constant_mintypmax_expression
stamptime_condition ::= mintypmax_expression
start_edge_offset ::= mintypmax_expression
threshold ::=constant_expression
timing_check_limit ::= expression
```

A.7.5.3 System timing check event definitions

```
timing_check_event ::=
      [timing_check_event_control] specify_terminal_descriptor [ &&& timing_check_condition ]
controlled_timing_check_event ::=
      timing_check_event_control specify_terminal_descriptor [ &&& timing_check_condition ]
timing_check_event_control ::=
      posedge
    | negedge
    | edge_control_specifier
specify_terminal_descriptor ::=
      specify_input_terminal_descriptor
    | specify_output_terminal_descriptor
edge_control_specifier ::= edge [ edge_descriptor [ , edge_descriptor ] ]
```


Copyright 2000 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

edge_descriptor[1] ::=
    01
  | **10**
  | z_or_x zero_or_one
  | zero_or_one z_or_x
zero_or_one ::= **0** | **1**
z_or_x ::= **x** | **X** | **z** | **Z**
timing_check_condition ::=
    scalar_timing_check_condition
  | **(** scalar_timing_check_condition **)**
scalar_timing_check_condition ::=
    expression
  | **~** expression
  | expression **==** scalar_constant
  | expression **===** scalar_constant
  | expression **!=** scalar_constant
  | expression **!==** scalar_constant
scalar_constant ::=
    1'b0 | **1'b1** | **1'B0** | **1'B1** | **'b0** | **'b1** | **'B0** | **'B1** | **1** | **0**

A.8 Expressions

A.8.1 Concatenations

genvar_concatenation ::= **{** expression { **,** expression } **}**
genvar_multiple_concatenation ::= **{** expression **{** expression { **,** expression } **} }**
net_concatenation ::= **{** net_concatenation_value { **,** net_concatenation_value } **}**
net_concatenation_value ::=
    hierarchical_net_identifier
  | hierarchical_net_identifier **[** expression **]** { **[** expression **]** }
  | hierarchical_net_identifier **[** expression **]** { **[** expression **]** } **[** range_expression **]**
  | hierarchical_net_identifier **[** range_expression **]**
  | net_concatenation
reg_concatenation ::= **{** reg_concatenation_value { **,** reg_concatenation_value } **}**
reg_concatenation_value ::=
    hierarchical_reg_identifier
  | hierarchical_reg_identifier **[** expression **]** { **[** expression **]** }
  | hierarchical_reg_identifier **[** expression **]** { **[** expression **]** } **[** range_expression **]**
  | hierarchical_reg_identifier **[** range_expression **]**
  | reg_concatenation
constant_concatenation ::= **{** constant_expression { **,** constant_expression } **}**
constant_multiple_concatenation ::= **{** constant_expression **{** constant_expression **} }**
concatenation ::= **{** expression { **,** expression } **}**
multiple_concatenation ::= **{** expression **{** expression { **,** expression } **} }**

Copyright 2000 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

A.8.2 Function calls

```
constant_function_call ::= function_identifier ( constant_expression { , constant_expression } )
function_call ::=
        hierarchical_function_identifier ( expression { , expression } ) [ attribute_instance ]
genvar_function_call ::= genvar_function_identifier ( genvar_expression { , genvar_expression } )
system_function_call ::= system_function_identifier [ ( expression { , expression } ) ]
```

A.8.3 Expressions

```
base_expression ::= expression
conditional_expression ::= expression1 ? expression2 : expression3
constant_expression ::=
        constant_primary
      | unary_operator constant_primary
      | constant_expression binary_operator constant_expression
      | conditional_expression
      | string
constant_mintypmax_expression ::=
        constant_expression
      | constant_expression : constant_expression : constant_expression
dimension_constant_expression ::= constant_expression
expression1 ::= constant_expression
expression2 ::= constant_expression
expression3 ::= constant_expression
expression ::=
        primary
      | unary_operator [ attribute_instance ] primary
      | expression binary_operator [ attribute_instance ] expression
      | expression ? [ attribute_instance ] expression : expression
      | string
genvar_expression ::=
        genvar_primary
      | unary_operator genvar_primary
      | genvar_expression binary_operator genvar_expression
      | genvar_expression ? genvar_expression  : genvar_expression
      | string
lsb_constant_expression ::= constant_expression
mintypmax_expression ::=
        expression
      | expression : expression : expression
msb_constant_expression ::= constant_expression
range_expression ::=
        msb_constant_expression : lsb_constant_expression
      | base_expression +: width_constant_expression
      | base_expression -: width_constant_expression
width_constant_expression ::= constant_expression
```


Copyright 2000 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

A.8.4 Primaries

constant_primary ::=
 number
| parameter_identifier
| specparam_identifier
| constant_concatenation
| constant_multiple_concatenation
| constant_function_call

genvar_primary ::=
 constant_primary
| genvar_identifier
| genvar_identifier [genvar_expression] { [genvar_expression] }
| genvar_identifier [genvar_expression] { [genvar_expression] } [range_expression]
| genvar_identifier [range_expression]
| genvar_concatenation
| genvar_multiple_concatenation
| genvar_function_call

primary ::=
 number
| hierarchical_identifier
| hierarchical_identifier [expression] { [expression] }
| hierarchical_identifier [expression] { [expression] } [range_expression]
| hierarchical_identifier [range_expression]
| concatenation
| multiple_concatenation
| function_call
| system_function_call
| constant_function_call
| (mintypmax_expression)

A.8.5 Expression left-side values

net_lvalue ::=
 hierarchical_net_identifier
| hierarchical_net_identifier [expression] { [expression] }
| hierarchical_net_identifier [expression] { [expression] } [range_expression]
| hierarchical_net_identifier [range_expression]
| net_concatenation

reg_lvalue ::=
 hierarchical_reg_identifier
| hierarchical_reg_identifier [expression] { [expression] }
| hierarchical_reg_identifier [expression] { [expression] } [range_expression]
| hierarchical_reg_identifier [range_expression]
| reg_concatenation

Copyright 2000 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

A.8.6 Operators

```
unary_operator ::=
      + | - | ! | ~ | & | ~& | | | ~| | ^ | ~^ | ^~
binary_operator ::=
      + | - | * | / | % | == | != | === | !== | && | || | **
    | < | <= | > | >= | & | | | ^ | ^~ | ~^ | >> | << | >>> | <<<
```

A.8.7 Numbers

```
number ::=
      decimal_number
    | octal_number
    | binary_number
    | hex_number
    | real_number
real_number[1] ::=
      unsigned_number . unsigned_number
    | unsigned_number [ . unsigned_number ] exp [ sign ] unsigned_number
exp ::= e | E
decimal_number ::=
      unsigned_number
    | [ size ] decimal_base unsigned_number
    | [ size ] decimal_base x_digit { _ }
    | [ size ] decimal_base z_digit { _ }
binary_number ::= [ size ] binary_base binary_value
octal_number ::= [ size ] octal_base octal_value
hex_number ::= [ size ] hex_base hex_value
sign ::= + | -
size ::= non_zero_unsigned_number
non_zero_unsigned_number[1] ::= non_zero_decimal_digit { _ | decimal_digit}
unsigned_number[1] ::= decimal_digit { _ | decimal_digit }
binary_value[1] ::= binary_digit { _ | binary_digit }
octal_value[1] ::= octal_digit { _ | octal_digit }
hex_value[1] ::= hex_digit { _ | hex_digit }
decimal_base[1] ::= '[s|S]d | '[s|S]D
binary_base[1] ::= '[s|S]b |  '[s|S]B
octal_base[1] ::= '[s|S]o | '[s|S]O
hex_base[1] ::= '[s|S]h | '[s|S]H
non_zero_decimal_digit ::= 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
decimal_digit ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
binary_digit ::= x_digit | z_digit | 0 | 1
octal_digit ::= x_digit | z_digit | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7
hex_digit ::=
      x_digit | z_digit | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
    | a | b | c | d | e | f | A | B | C | D | E | F
```


Copyright 2000 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

x_digit ::= **x** | **X**
z_digit ::= **z** | **Z** | **?**

A.8.8 Strings

string ::= **"** { Any_ASCII_Characters_except_new_line } **"**

A.9 General

A.9.1 Attributes

attribute_instance ::= **(*** attr_spec { **,** attr_spec } **; *)**
attr_spec ::=
    attr_name **=** constant_expression
  | attr_name
attr_name ::= { Any_ASCII_Characters_except_new_line }

A.9.2 Comments

comment ::=
    one_line_comment
  | block_comment
one_line_comment ::= **//** comment_text \n
block_comment ::= **/*** comment_text ***/**
comment_text ::= { Any_ASCII_character }

A.9.3 Identifiers

arrayed_identifier ::=
    simple_arrayed_identifier
  | escaped_arrayed_identifier
block_identifier ::= identifier
cell_identifier ::= identifier
config_identifier ::= identifier
edge_identifier ::= identifier
escaped_arrayed_identifier ::= escaped_identifier [[range]]
escaped_hierarchical_identifier[4] ::=
    escaped_hierarchical_branch
      [{ .simple_hierarchical_branch | .escaped_hierarchical_branch }]
escaped_identifier ::= **** {Any_ASCII_character_except_white_space} white_space
event_identifier ::= identifier
function_identifier ::= identifier
gate_instance_identifier ::= arrayed_identifier
generate_block_identifier ::= identifier
genvar_function_identifier ::= identifier **/*** Hierarchy disallowed ***/**
genvar_identifier ::= identifier
hierarchical_block_identifier ::= hierarchical_identifier
hierarchical_event_identifier ::= hierarchical_identifier

Copyright 2000 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

hierarchical_function_identifier ::= hierarchical_identifier
hierarchical_identifier ::=
    simple_hierarchical_identifier
  | escaped_hierarchical_identifier
hierarchical_net_identifier ::= hierarchical_identifier
hierarchical_reg_identifier ::= hierarchical_identifier
hierarchical_task_identifier ::= hierarchical_identifier
identifier ::=
    simple_identifier
  | escaped_identifier
inout_port_identifier ::= identifier
input_identifier ::= identifier
input_port_identifier ::= identifier
instance_identifier ::= identifier
library_identifier ::= identifier
memory_identifier ::= identifier
module_identifier ::= identifier
module_instance_identifier ::= arrayed_identifier
net_identifier ::= identifier
output_identifier ::= identifier
output_port_identifier ::= identifier
parameter_identifier ::= identifier
port_identifier ::= identifier
real_identifier ::= identifier
register_identifier ::= identifier
simple_arrayed_identifier ::= simple_identifier [range]
simple_hierarchical_identifier[3] ::=
    simple_hierarchical_branch [**.**escaped_identifier]
simple_identifier[2] ::= [**a-zA-Z_**] { [**a-zA-Z0-9_$**] }
specparam_identifier ::= identifier
system_function_identifier[5] ::= **$**[**a-zA-Z0-9_$**]{ [**a-zA-Z0-9_$**] }
system_task_identifier[5] ::= **$**[**a-zA-Z0-9_$**]{ [**a-zA-Z0-9_$**] }
task_identifier ::= identifier
terminal_identifier ::= identifier
text_macro_identifier ::= simple_identifier
topmodule_identifier ::= identifier
udp_identifier ::= identifier
udp_instance_identifier ::= arrayed_identifier

A.9.4 Identifier branches

simple_hierarchical_branch[3] ::=
    simple_identifier [[unsigned_number]]
      [{ **.**simple_identifier [[unsigned_number]] }]
escaped_hierarchical_branch[4] ::=
    escaped_identifier [[unsigned_number]]
      [{ **.**escaped_identifier [[unsigned_number]] }]

Copyright 2000 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

A.9.5 White space

white_space ::= space | tab | newline | eof[6]

NOTES

1) Embedded spaces are illegal.
2) A `simple_identifier` and `arrayed_reference` shall start with an alpha or underscore (_) character, shall have at least one character, and shall not have any spaces.
3) The period (`.`) in `simple_hierarchical_identifier` and `simple_hierarchical_branch` shall not be preceded or followed by `white_space.`
4) The period in `escaped_hierarchical_identifier` and `escaped_hierarchical_branch` shall be preceded by `white_space,` but shall not be followed by `white_space`.
5) The `$` character in a `system_function_identifier` or `system_task_identifier` shall not be followed by white_space. A `system_function_identifier` or `system_task_identifier` shall not be escaped.
6) End of file.

Copyright 2000 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

Annex B

List of keywords

(normative)

B.1 All keywords

Keywords are predefined nonescaped identifiers that define Verilog language constructs. An escaped identifier shall not be treated as a keyword.

Copyright 2000 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

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

B.2 Configuration

Configuration keywords are predefined nonescaped identifiers that define and are only applicable to Verilog configuration constructs. All configuration keywords are used between the Verilog keywords **config** and **endconfig**. An escaped identifier shall not be treated as a configuration keyword.

Copyright 2000 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

design
instance
cell
use
liblist

B.3 Library

Library keywords are predefined nonescaped identifiers that define and are only applicable to Verilog library files. A library file is not part of the Verilog design and configuration input stream.

include
incdir

Copyright 2000 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

 Copyright 2000 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

Annex C

System tasks and functions

(informative)

The system tasks and functions described in this annex are for informative purposes only and are not part of the IEEE standard Verilog HDL.

This annex describes system tasks and functions as companions to the system tasks and functions described in Section 17. The system tasks and functions described in this annex may not be available in all implementations of the Verilog HDL. The following system tasks and functions are described in this annex:

The word *tool* in this annex refers to an implementation of Verilog HDL, typically a logic simulator.

C.1 $countdrivers

Syntax:

$countdrivers (net, [net_is_forced, number_of_01x_drivers, number_of_0_drivers,
number_of_1_drivers, number_of_x_drivers]);

The **$countdrivers** system function is provided to count the number of drivers on a specified net so that bus contention can be identified.

This system function returns a `0` if there is no more than one driver on the net and returns a `1` otherwise (indicating contention). The specified net shall be a scalar or a bit-select of a vector net. The number of parameters to the system function may vary according to how much information is desired.

Copyright 2000 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

If additional parameters are supplied to the **$countdrivers** function, each parameter returns the information described in Table C1.

Table C1—Parameter return value for $countdriver function

Parameter	Return value
net_is_forced	1 if net is forced 0 otherwise
number_of_01x_drivers	An integer representing the number of drivers on the net that are in 0, 1, or x state. This represents the total number of drivers that are not forced
number_of_0_drivers	An integer representing the number of drivers on the net that are in `0` state
number_of_1_drivers	An integer representing the number of drivers on the net that are in `1` state
number_of_x_drivers	An integer representing the number of drivers on the net that are in `x` state

C.2 $getpattern

Syntax:

$getpattern `( mem_element );`

The system function **$getpattern** provides for fast processing of stimulus patterns that have to be propagated to a large number of scalar inputs. The function reads stimulus patterns that have been loaded into a memory using the **$readmemb** or **$readmemh** system tasks.

Use of this function is limited, however: it may only be used in a continuous assignment statement where the lefthand side is a concatenation of scalar nets, and the parameter to the system function is a memory element reference.

Example:

The following example shows how stimuli stored in a file can be read into a memory using **$readmemb** and applied to the circuit one pattern at a time using **$getpattern**.

The memory `in_mem` is initialized with the stimulus patterns by the **$readmemb** task. The integer variable `index` selects which pattern is being applied to the circuit. The `for` loop increments the integer variable `index` periodically to sequence the patterns.

 Copyright 2000 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

```
module top;
parameter in_width=10,
          patterns=200,
          step=20;
reg [1:in_width] in_mem[1:patterns];
integer index;

// declare scalar inputs
wire i1,i2,i3,i4,i5,i6,i7,i8,i9,i10;

// assign patterns to circuit scalar inputs (a new pattern
// is applied to the circuit each time index changes value)
assign {i1,i2,i3,i4,i5,i6,i7,i8,i9,i10} = $getpattern(in_mem[index]);
initial begin
   // read stimulus patterns into memory
   $readmemb("patt.mem", in_mem);

   // step through patterns (note that each assignment
   // to index will drive a new pattern onto the circuit
   // inputs from the $getpattern system task specified above
   for (index = 1; index <= patterns; index = index + 1)
      #step;
end

   // instantiate the circuit module - e.g.
mod1 cct (o1,o2,o3,o4,o5, i1,i2,i3,i4,i5,i6,i7,i8,i9,i10);

endmodule
```

C.3 $input

Syntax:

$input ("*filename*");

The **$input** system task allows command input text to come from a named file instead of from the terminal. At the end of the command file, the input is switched back to the terminal.

C.4 $key and $nokey

Syntax:

$key [(*"filename*")] ;
$nokey ;

A key file is created whenever interactive mode is entered for the first time during simulation. The key file contains all of the text that has been typed in from the standard input. The file also contains information about asynchronous interrupts.

Copyright 2000 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

The **$nokey** and **$key** system tasks are used to disable and re-enable output to the key file. An optional file name parameter for **$key** causes the old key file to be closed, a new file to be created, and output to be directed to the new file.

C.5 $list

Syntax:

$list [(hierarchical_name)] ;

When invoked without a parameter, **$list** produces a listing of the module, task, function, or named block that is defined as the current scope setting. If an optional parameter is supplied, it shall refer to a specific module, task, function or named block, in which case the specified object is listed.

C.6 $log and $nolog

Syntax:

$log [(*"filename*")] ;
$nolog ;

A log file contains a copy of all the text that is printed to the standard output. The log file may also contain, at the beginning of the file, the host command that was used to run the tool.

The **$nolog** and **$log** system tasks are used to disable and re-enable output to the log file. The **$nolog** task disables output to the log file, while the **$log** task re-enables the output. An optional file name parameter for **$log** causes the old file to be closed, a new log file to be created, and output to be directed to the new log file.

C.7 $reset, $reset_count, and $reset_value

Syntax:

$reset [(stop_value [, reset_value , [diagnostics_value]])] ;
$reset_count ;
$reset_value ;

The **$reset** system task enables a tool to be reset to its “Time 0” state so that processing (e.g., simulation) can begin again.

The **$reset_count** system function keeps track of the number of times the tool is reset. The **$reset_value** system function returns the value specified by the `reset_value` parameter argument to the **$reset** system task. The **$reset_value** system function is used to communicate information from before a reset of a tool to the time 0 state to after the reset.

The following are some of the simulation methods that can be employed with this system task and these system functions:

— Determine the **force** statements a design needs to operate correctly, reset the simulation time to 0, enter these **force** statements, and start to simulate again
— Reset the simulation time to 0 and apply new stimuli
— Determine that debug system tasks, such as **$monitor** and **$strobe**, are keeping track of the correct nets or regs, reset the simulation time to 0, and begin simulation again

 Copyright 2000 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

The **$reset** system task tells a tool to return the processing of the design to its logical state at time 0. When a tool executes the **$reset** system task, it takes the following actions to stop the process:

1) Disables all concurrent activity, initiated in either **initial** and **always** procedural blocks in the source description or through interactive mode (disables, for example, all **force** and **assign** statements, the current **$monitor** system task, and any other active tasks)

2) Cancels all scheduled simulation events

After a simulation tool executes the **$reset** system task, the simulation is in the following state:

— The simulation time is `0`.
— All regs and nets contain their initial values.
— The tool begins to execute the first procedural statements in all **initial** and **always** blocks.

The `stop_value` argument indicates whether interactive mode or processing is entered immediately after resetting of the tool. A value of `0` or no argument causes interactive mode to be entered after resetting the tool. A nonzero value passed to **$reset** causes the tool to begin processing immediately.

The `reset_value` argument is an integer that specifies whose value is returned by the **$reset_value** system function after the tool is reset. All declared integers return to their initial value after reset, but entering an integer as this argument allows access to what its value was before the reset with the **$reset_value** system function. This argument provides a means of communicating information from before the reset of a tool to after the reset of the tool.

The `diagnostic_value` specifies the kind of diagnostic messages a tool displays before it resets the time to `0`. Increasing integer values results in increased information. A value of zero results in no diagnostic message.

C.8 $save, $restart, and $incsave

Three system tasks **$save**, **$restart**, and **$incsave** work in conjunction with one another to save the complete state of simulation into a permanent file such that the simulation state can be reloaded at a later time and processing can continue where it left off.

Syntax:

$save("*file_name*");
$restart("*file_name*");
$incsave("*incremental_file_name*");

All three system tasks take a file name as a parameter. The file name has to be supplied as a string enclosed in quotation marks.

The **$save** system task saves the complete state into the host operating system file specified as a parameter.

The **$incsave** system task saves only what has changed since the last invocation of **$save**. It is not possible to do an incremental save on any file other than the one produced by the last **$save**.

The **$restart** system task restores a previously saved state from a specified file.

Restarting from an incremental save is similar to restarting from a full save, except that the name of the incremental save file is specified in the restart command. The full save file that the incremental save file was based upon shall still be present, as it is required for a successful restart. If the full save file has been changed in any way since the incremental save was performed, errors will result.

Copyright 2000 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

The incremental restart is useful for going back in time. If a full save is performed near the beginning of processing, and an incremental save is done at regular intervals, then going back in time is as simple as restarting from the appropriate file.

Example:

```
module checkpoint;

initial
   #500 $save("save.dat");  // full save

always begin         // incremental save every 10000 units,
                     // files are recycled every 40000 units
   #100000 $incsave("inc1.dat");
   #100000 $incsave("inc2.dat");
   #100000 $incsave("inc3.dat");
   #100000 $incsave("inc4.dat");
end
endmodule
```

C.9 $scale

Syntax:

$scale (hierarchical_name) ;

The **$scale** function takes a time value from a module with one time unit to be used in a module with a different time unit. The time value is converted from the time unit of one module to the time unit of the module that invokes **$scale**.

C.10 $scope

Syntax:

$scope (hierarchical_name) ;

The **$scope** system task allows a particular level of hierarchy to be specified as the scope for identifying objects. This task accepts a single parameter argument that shall be the complete hierarchical name of a module, task, function, or named block. The initial setting of the interactive scope is the first top-level module.

C.11 $showscopes

Syntax:

$showscopes [(n)];

The **$showscopes** system task produces a complete list of modules, tasks, functions, and named blocks that are defined *at the current scope level*. An optional integer parameter can be given to **$showscopes**. A nonzero parameter value causes all the modules, tasks, functions, and named blocks in or below the current hierarchical scope to be listed. No parameter or a zero value results in only objects at the current scope level to be listed.

Copyright 2000 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

C.12 $showvars

Syntax:

$showvars [(list_of_variables)] ;

The **$showvars** system task produces status information for reg and net variables, both scalar and vector. When invoked without parameters, **$showvars** displays the status of all variables in the current scope. When invoked with a list of variables, **$showvars** shows only the status of the specified variables. If the list of variables includes a bit-select or part-select of a reg or net, then the status information for all the bits of that reg or net are displayed.

C.13 $sreadmemb and $sreadmemh

Syntax:

$sreadmemb (mem_name , start_address , finish_address , string { , string }) ;
$sreadmemh (mem_name , start_address , finish_address , string { , string }) ;

The system tasks **$sreadmemb** and **$sreadmemh** load data into memory `mem_name` from a character string.

The **$sreadmemh** and **$sreadmemb** system tasks take memory data values and addresses as string arguments. The start and finish addresses indicate the bounds for where the data from strings will be stored in the memory. These strings take the same format as the strings that appear in the input files passed as arguments to **$readmemb** and **$readmemh**.

Copyright 2000 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

 Copyright 2000 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

Annex D

Compiler directives

(informative)

The compiler directives described in this annex are for informative purposes only and are not part of the IEEE standard Verilog HDL.

This annex describes additional compiler directives as companions to the compiler directives described in Section 19. The compiler directives described in this annex may not be available in all implementations of the Verilog HDL. The following compiler directives are described in this annex:

The word *tool* in this annex refers to an implementation of Verilog HDL, typically a logic simulator.

D.1 `default_decay_time

The **`default_decay_time** compiler directives specifies the decay time for the trireg nets that do not have any decay time specified in the declaration. This compiler directive applies to all of the trireg nets in all the modules that follow it in the source description. An argument specifying the charge decay time shall be used with this compiler directive.

Syntax:

`default_decay_time*integer*_constant | *real*_constant | **infinite**

Examples:

Example 1—The following example shows how the default decay time for all trireg nets can be set to 100 time units:

`default_decay_time100

Example 2—The following example shows how to avoid charge decay on trireg nets:

`default_decay_timeinfinite

The keyword infinite specifies no charge decay for all the trireg nets that do not have decay time specification.

D.2 `default_trireg_strength

The **`default_trireg_strength** compiler directive specifies the charge strength of **trireg** nets.

Syntax:

Copyright 2000 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

`default_trireg_strength*integer*_constant

The integer constant shall be between 0 and 250. It indicates the relative strength of the capacitance on the trireg net.

D.3 `delay_mode_distributed

The **`delay_mode_distributed** compiler directive specifies the distributed delay mode for all modules that follow this directive in the source description.

Syntax:

`delay_mode_distributed

This compiler directive shall be used before the declaration of the module whose delay mode is being controlled.

D.4 `delay_mode_path

The **`delay_mode_path** compiler directive specifies the path delay mode for all modules that follow this directive in the source description.

Syntax:

`delay_mode_path

This compiler directive shall be used before the declaration of the module whose delay mode is being controlled.

D.5 `delay_mode_unit

The **`delay_mode_unit** compiler directive specifies the unit delay mode for all modules that follow this directive in the source description.

Syntax:

`delay_mode_unit

This compiler directive shall be used before the declaration of the module whose delay mode is being controlled.

D.6 `delay_mode_zero

The **`delay_mode_zero** compiler directive specifies the zero-delay mode for all modules that follow this directive in the source description.

Syntax:

`delay_mode_zero

This compiler directive shall be used before the declaration of the module whose delay mode is being controlled.

 Copyright 2000 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

Annex E

acc_user.h

```
/****************************************************************************
 * acc_user.h
 *
 * IEEE 1364-2000 Verilog HDL Programming Language Interface (PLI).
 *
 * This file contains the constant definitions, structure definitions, and
 * routine declarations for the Verilog Programming Language Interface ACC
 * access routines.
 *
 ****************************************************************************/

#ifndef ACC_USER_H
#define ACC_USER_H

/*--------------------------------------------------------------------------*/
/*---------------------------- Portability Help ----------------------------*/
/*--------------------------------------------------------------------------*/
/* Sized variables */
#ifndef PLI_TYPES
#define PLI_TYPES
typedef int             PLI_INT32;
typedef unsigned int    PLI_UINT32;
typedef short           PLI_INT16;
typedef unsigned short  PLI_UINT16;
typedef char            PLI_BYTE8;
typedef unsigned char   PLI_UBYTE8;
#endif

/* export a symbol */
#if WIN32
#ifndef PLI_DLLISPEC
#define PLI_DLLISPEC __declspec(dllimport)
#define ACC_USER_DEFINED_DLLISPEC 1
#endif
#else
#ifndef PLI_DLLISPEC
#define PLI_DLLISPEC
#endif
#endif

/* import a symbol */
#if WIN32
#ifndef PLI_DLLESPEC
#define PLI_DLLESPEC __declspec(dllexport)
#define ACC_USER_DEFINED_DLLESPEC 1
#endif
#else
#ifndef PLI_DLLESPEC
#define PLI_DLLESPEC
#endif
#endif

/* mark a function as external */
#ifndef PLI_EXTERN
#define PLI_EXTERN
#endif

/* mark a variable as external */
#ifndef PLI_VEXTERN
#define PLI_VEXTERN extern
```


Copyright 2000 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

```
#endif

#ifndef PLI_PROTOTYPES
#define PLI_PROTOTYPES
#define PROTO_PARAMS(params) params
/* object is imported by the application */
#define XXTERN PLI_EXTERN PLI_DLLISPEC
/* object is exported by the application */
#define EETERN PLI_EXTERN PLI_DLLESPEC
#endif

/*
 * The following group of defines exists purely for backwards compatibility
 */
#ifndef PLI_EXTRAS
#define PLI_EXTRAS
#define bool                    int
#define true                    1
#define TRUE                    1
#define false                   0
#define FALSE                   0
#define null                    0L
#endif


/*---------------------------------------------------------------------------*/
/*------------------------------ definitions --------------------------------*/
/*---------------------------------------------------------------------------*/

/*----------------------------- general defines -----------------------------*/
typedef PLI_INT32   *HANDLE;
#ifndef VPI_USER_CDS_H
typedef PLI_INT32   *handle;
#endif

/*------------------------------ object types -------------------------------*/
#define    accModule               20
#define    accScope                21
#define    accNet                  25
#define    accReg                  30
#define    accRegister             accReg
#define    accPort                 35
#define    accTerminal             45
#define    accInputTerminal        46
#define    accOutputTerminal       47
#define    accInoutTerminal        48
#define    accCombPrim            140
#define    accSeqPrim             142
#define    accAndGate             144
#define    accNandGate            146
#define    accNorGate             148
#define    accOrGate              150
#define    accXorGate             152
#define    accXnorGate            154
#define    accBufGate             156
#define    accNotGate             158
#define    accBufif0Gate          160
#define    accBufif1Gate          162
#define    accNotif0Gate          164
#define    accNotif1Gate          166
#define    accNmosGate            168
#define    accPmosGate            170
#define    accCmosGate            172
#define    accRnmosGate           174
#define    accRpmosGate           176
#define    accRcmosGate           178
#define    accRtranGate           180
#define    accRtranif0Gate        182
#define    accRtranif1Gate        184
#define    accTranGate            186
#define    accTranif0Gate         188
#define    accTranif1Gate         190
```

 Copyright 2000 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

```
#define    accPullupGate          192
#define    accPulldownGate        194
#define    accIntegerParam        200
#define    accIntParam            accIntegerParam
#define    accRealParam           202
#define    accStringParam         204
#define    accTchk                208
#define    accPrimitive           210
#define    accBit                 212
#define    accPortBit             214
#define    accNetBit              216
#define    accRegBit              218
#define    accParameter           220
#define    accSpecparam           222
#define    accTopModule           224
#define    accModuleInstance      226
#define    accCellInstance        228
#define    accModPath             230
#define    accInterModPath        236
#define    accScalarPort          250
#define    accBitSelectPort       252
#define    accPartSelectPort      254
#define    accVectorPort          256
#define    accConcatPort          258
#define    accWire                260
#define    accWand                261
#define    accWor                 262
#define    accTri                 263
#define    accTriand              264
#define    accTrior               265
#define    accTri0                266
#define    accTri1                267
#define    accTrireg              268
#define    accSupply0             269
#define    accSupply1             270
#define    accNamedEvent          280
#define    accEventVar            accNamedEvent
#define    accIntegerVar          281
#define    accIntVar              281
#define    accRealVar             282
#define    accTimeVar             283
#define    accScalar              300
#define    accVector              302
#define    accExpandedVector      306
#define    accUnExpandedVector    307
#define    accProtected           308
#define    accSetup               366
#define    accHold                367
#define    accWidth               368
#define    accPeriod              369
#define    accRecovery            370
#define    accSkew                371
#define    accNochange            376
#define    accNoChange            accNochange
#define    accSetuphold           377
#define    accInput               402
#define    accOutput              404
#define    accInout               406
#define    accMixedIo             407
#define    accPositive            408
#define    accNegative            410
#define    accUnknown             412
#define    accPathTerminal        420
#define    accPathInput           422
#define    accPathOutput          424
#define    accDataPath            426
#define    accTchkTerminal        428
#define    accBitSelect           500
#define    accPartSelect          502
#define    accTask                504
#define    accFunction            506
#define    accStatement           508
```


Copyright 2000 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

```
#define    accTaskCall            510
#define    accFunctionCall        512
#define    accSystemTask          514
#define    accSystemFunction      516
#define    accSystemRealFunction  518
#define    accUserTask            520
#define    accUserFunction        522
#define    accUserRealFunction    524
#define    accConstant            600
#define    accConcat              610
#define    accOperator            620
#define    accMinTypMax           696

/*------------------ parameter values for acc_configure() --------------------*/
#define    accPathDelayCount        1
#define    accPathDelimStr          2
#define    accDisplayErrors         3
#define    accDefaultAttr0          4
#define    accToHiZDelay            5
#define    accEnableArgs            6
#define    accDisplayWarnings       8
#define    accDevelopmentVersion   11
#define    accMapToMipd            17
#define    accMinTypMaxDelays      19

/*------------ edge information used by acc_handle_tchk(), etc.  ------------*/
#define accNoedge                   0
#define accNoEdge                   0
#define accEdge01                   1
#define accEdge10                   2
#define accEdge0x                   4
#define accEdgex1                   8
#define accEdge1x                  16
#define accEdgex0                  32
#define accPosedge                 13
#define accPosEdge                 accPosedge
#define accNegedge                 50
#define accNegEdge                 accNegedge

/*------------------------------- delay modes --------------------------------*/
#define accDelayModeNone            0
#define accDelayModePath            1
#define accDelayModeDistrib         2
#define accDelayModeUnit            3
#define accDelayModeZero            4
#define accDelayModeVeritime        5

/*------------ values for type field in t_setval_delay structure ------------*/
#define accNoDelay                  0
#define accInertialDelay            1
#define accTransportDelay           2
#define accPureTransportDelay       3
#define accForceFlag                4
#define accReleaseFlag              5
#define accAssignFlag               6
#define accDeassignFlag             7

/*------------ values for type field in t_setval_value structure ------------*/
#define accBinStrVal                1
#define accOctStrVal                2
#define accDecStrVal                3
#define accHexStrVal                4
#define accScalarVal                5
#define accIntVal                   6
#define accRealVal                  7
#define accStringVal                8
#define accVectorVal               10

/*------------------------------ scalar values -------------------------------*/
#define acc0                        0
#define acc1                        1
#define accX                        2
```

 Copyright 2000 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

```
#define accZ                        3

/*---------------------------- VCL scalar values ----------------------------*/
#define vcl0                        acc0
#define vcl1                        acc1
#define vclX                        accX
#define vclZ                        accZ

/*----------- values for vc_reason field in t_vc_record structure -----------*/
#define logic_value_change          1
#define strength_value_change       2
#define real_value_change           3
#define vector_value_change         4
#define event_value_change          5
#define integer_value_change        6
#define time_value_change           7
#define sregister_value_change      8
#define vregister_value_change      9
#define realtime_value_change      10

/*---------------------------- VCL strength values ----------------------------*/
#define vclSupply                   7
#define vclStrong                   6
#define vclPull                     5
#define vclLarge                    4
#define vclWeak                     3
#define vclMedium                   2
#define vclSmall                    1
#define vclHighZ                    0

/*----------------------- flags used with acc_vcl_add -----------------------*/
#define vcl_verilog_logic           2
#define VCL_VERILOG_LOGIC           vcl_verilog_logic
#define vcl_verilog_strength        3
#define VCL_VERILOG_STRENGTH        vcl_verilog_strength

/*---------------------- flags used with acc_vcl_delete ---------------------*/
#define vcl_verilog                 vcl_verilog_logic
#define VCL_VERILOG                 vcl_verilog

/*---------- values for the type field in the t_acc_time structure --------- */
#define accTime                     1
#define accSimTime                  2
#define accRealTime                 3

/*------------------------------ product types ------------------------------*/
#define accSimulator                1
#define accTimingAnalyzer           2
#define accFaultSimulator           3
#define accOther                    4


/*---------------------------------------------------------------------------*/
/*-------------------------- structure definitions --------------------------*/
/*---------------------------------------------------------------------------*/

typedef PLI_INT32 (*consumer_function)();

/*----------------- data structure used with acc_set_value() ----------------*/
typedef struct t_acc_time
{
  PLI_INT32 type;
  PLI_INT32 low,
            high;
  double    real;
} s_acc_time, *p_acc_time;

/*----------------- data structure used with acc_set_value() ----------------*/
typedef struct t_setval_delay
{
  s_acc_time time;
  PLI_INT32  model;
```


Copyright 2000 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

```
} s_setval_delay, *p_setval_delay;

/*--------------------- data structure of vector values ---------------------*/
typedef struct t_acc_vecval
{
  PLI_INT32 aval;
  PLI_INT32 bval;
} s_acc_vecval, *p_acc_vecval;

/*------ data structure used with acc_set_value() and acc_fetch_value() -----*/
typedef struct t_setval_value
{
  PLI_INT32 format;
  union
    {
      PLI_BYTE8    *str;
      PLI_INT32     scalar;
      PLI_INT32     integer;
      double        real;
      p_acc_vecval  vector;
    } value;
} s_setval_value, *p_setval_value, s_acc_value, *p_acc_value;

/*----------------------- structure for VCL strengths -----------------------*/
typedef struct t_strengths
{
  PLI_UBYTE8 logic_value;
  PLI_UBYTE8 strength1;
  PLI_UBYTE8 strength2;
} s_strengths, *p_strengths;

/*--------------- structure passed to callback routine for VCL --------------*/
typedef struct t_vc_record
{
  PLI_INT32  vc_reason;
  PLI_INT32  vc_hightime;
  PLI_INT32  vc_lowtime;
  PLI_BYTE8 *user_data;
  union
    {
      PLI_UBYTE8  logic_value;
      double      real_value;
      handle      vector_handle;
      s_strengths strengths_s;
    } out_value;
} s_vc_record, *p_vc_record;

/*------------- structure used with acc_fetch_location() routine ------------*/
typedef struct t_location
{
  PLI_INT32  line_no;
  PLI_BYTE8 *filename;
} s_location, *p_location;

/*---------- structure used with acc_fetch_timescale_info() routine ---------*/
typedef struct t_timescale_info
{
  PLI_INT16 unit;
  PLI_INT16 precision;
} s_timescale_info, *p_timescale_info;


/*--------------------------------------------------------------------------*/
/*-------------------------- routine declarations --------------------------*/
/*--------------------------------------------------------------------------*/

XXTERN PLI_INT32   acc_append_delays PROTO_PARAMS((handle object, ...));
XXTERN PLI_INT32   acc_append_pulsere PROTO_PARAMS((handle object, double val1r, double
val1x, ...));
XXTERN void        acc_close PROTO_PARAMS((void));
XXTERN handle     *acc_collect PROTO_PARAMS((handle (*p_next_routine)(), handle
scope_object, PLI_INT32 *aof_count));
```


Copyright 2000 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

```
XXTERN PLI_INT32   acc_compare_handles PROTO_PARAMS((handle h1, handle h2));
XXTERN PLI_INT32   acc_configure PROTO_PARAMS((PLI_INT32 item, PLI_BYTE8 *value));
XXTERN PLI_INT32   acc_count PROTO_PARAMS((handle (*next_func)(), handle
object_handle));
XXTERN PLI_INT32   acc_fetch_argc PROTO_PARAMS((void));
XXTERN PLI_BYTE8 **acc_fetch_argv PROTO_PARAMS((void));
XXTERN double      acc_fetch_attribute PROTO_PARAMS((handle object, ...));
XXTERN PLI_INT32   acc_fetch_attribute_int PROTO_PARAMS((handle object, ...));
XXTERN PLI_BYTE8  *acc_fetch_attribute_str PROTO_PARAMS((handle object, ...));
XXTERN PLI_BYTE8  *acc_fetch_defname PROTO_PARAMS((handle object_handle));
XXTERN PLI_INT32   acc_fetch_delay_mode PROTO_PARAMS((handle object_p));
XXTERN PLI_INT32   acc_fetch_delays PROTO_PARAMS((handle object, ...));
XXTERN PLI_INT32   acc_fetch_direction PROTO_PARAMS((handle object_handle));
XXTERN PLI_INT32   acc_fetch_edge PROTO_PARAMS((handle acc_obj));
XXTERN PLI_BYTE8  *acc_fetch_fullname PROTO_PARAMS((handle object_handle));
XXTERN PLI_INT32   acc_fetch_fulltype PROTO_PARAMS((handle object_h));
XXTERN PLI_INT32   acc_fetch_index PROTO_PARAMS((handle object_handle));
XXTERN double      acc_fetch_itfarg PROTO_PARAMS((PLI_INT32 n, handle tfinst));
XXTERN PLI_INT32   acc_fetch_itfarg_int PROTO_PARAMS((PLI_INT32 n, handle tfinst));
XXTERN PLI_BYTE8  *acc_fetch_itfarg_str PROTO_PARAMS((PLI_INT32 n, handle tfinst));
XXTERN PLI_INT32   acc_fetch_location PROTO_PARAMS((p_location location_p, handle
object));
XXTERN PLI_BYTE8  *acc_fetch_name PROTO_PARAMS((handle object_handle));
XXTERN PLI_INT32   acc_fetch_paramtype PROTO_PARAMS((handle param_p));
XXTERN double      acc_fetch_paramval PROTO_PARAMS((handle param));
XXTERN PLI_INT32   acc_fetch_polarity PROTO_PARAMS((handle path));
XXTERN PLI_INT32   acc_fetch_precision PROTO_PARAMS((void));
XXTERN PLI_INT32  acc_fetch_pulsere PROTO_PARAMS((handle path_p, double *val1r, double
*val1e, ...));
XXTERN PLI_INT32   acc_fetch_range PROTO_PARAMS((handle node, PLI_INT32 *msb, PLI_INT32
*lsb));
XXTERN PLI_INT32   acc_fetch_size PROTO_PARAMS((handle obj_h));
XXTERN double      acc_fetch_tfarg PROTO_PARAMS((PLI_INT32 n));
XXTERN PLI_INT32   acc_fetch_tfarg_int PROTO_PARAMS((PLI_INT32 n));
XXTERN PLI_BYTE8  *acc_fetch_tfarg_str PROTO_PARAMS((PLI_INT32 n));
XXTERN void        acc_fetch_timescale_info PROTO_PARAMS((handle obj, p_timescale_info
aof_timescale_info));
XXTERN PLI_INT32   acc_fetch_type PROTO_PARAMS((handle object_handle));
XXTERN PLI_BYTE8  *acc_fetch_type_str PROTO_PARAMS((PLI_INT32 type));
XXTERN PLI_BYTE8  *acc_fetch_value PROTO_PARAMS((handle object_handle, PLI_BYTE8
*format_str, p_acc_value acc_value_p));
XXTERN void        acc_free PROTO_PARAMS((handle *array_ptr));
XXTERN handle      acc_handle_by_name PROTO_PARAMS((PLI_BYTE8 *inst_name, handle
scope_p));
XXTERN handle      acc_handle_condition PROTO_PARAMS((handle obj));
XXTERN handle      acc_handle_conn PROTO_PARAMS((handle term_p));
XXTERN handle      acc_handle_datapath PROTO_PARAMS((handle path));
XXTERN handle      acc_handle_hiconn PROTO_PARAMS((handle port_ref));
XXTERN handle      acc_handle_interactive_scope PROTO_PARAMS((void));
XXTERN handle      acc_handle_itfarg PROTO_PARAMS((PLI_INT32 n, void *suena_inst));
XXTERN handle      acc_handle_loconn PROTO_PARAMS((handle port_ref));
XXTERN handle      acc_handle_modpath PROTO_PARAMS((handle mod_p, PLI_BYTE8
*pathin_name, PLI_BYTE8 *pathout_name, ...));
XXTERN handle      acc_handle_notifier PROTO_PARAMS((handle tchk));
XXTERN handle      acc_handle_object PROTO_PARAMS((PLI_BYTE8 *inst_name, ...));
XXTERN handle      acc_handle_parent PROTO_PARAMS((handle object_p));
XXTERN handle      acc_handle_path PROTO_PARAMS((handle source, handle destination));
XXTERN handle      acc_handle_pathin PROTO_PARAMS((handle path_p));
XXTERN handle      acc_handle_pathout PROTO_PARAMS((handle path_p));
XXTERN handle     acc_handle_port PROTO_PARAMS((handle mod_handle, PLI_INT32 port_num,
...));
XXTERN handle      acc_handle_scope PROTO_PARAMS((handle object));
XXTERN handle      acc_handle_simulated_net PROTO_PARAMS((handle net_h));
XXTERN handle      acc_handle_tchk PROTO_PARAMS((handle mod_p, PLI_INT32 tchk_type,
PLI_BYTE8 *arg1_conn_name, PLI_INT32 arg1_edgetype, ...));
XXTERN handle      acc_handle_tchkarg1 PROTO_PARAMS((handle tchk));
XXTERN handle      acc_handle_tchkarg2 PROTO_PARAMS((handle tchk));
XXTERN handle      acc_handle_terminal PROTO_PARAMS((handle gate_handle, PLI_INT32
terminal_index));
XXTERN handle      acc_handle_tfarg PROTO_PARAMS((PLI_INT32 n));
XXTERN handle      acc_handle_tfinst PROTO_PARAMS((void));
XXTERN PLI_INT32   acc_initialize PROTO_PARAMS((void));
```


Copyright 2000 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

```
XXTERN handle      acc_next PROTO_PARAMS((PLI_INT32 *type_list, handle h_scope, handle
h_object));
XXTERN handle      acc_next_bit PROTO_PARAMS ((handle vector, handle bit));
XXTERN handle      acc_next_cell PROTO_PARAMS((handle scope, handle cell));
XXTERN handle      acc_next_cell_load PROTO_PARAMS((handle net_handle, handle load));
XXTERN handle      acc_next_child PROTO_PARAMS((handle mod_handle, handle child));
XXTERN handle      acc_next_driver PROTO_PARAMS((handle net, handle driver));
XXTERN handle      acc_next_hiconn PROTO_PARAMS((handle port, handle hiconn));
XXTERN handle      acc_next_input PROTO_PARAMS((handle path, handle pathin));
XXTERN handle      acc_next_load PROTO_PARAMS((handle net, handle load));
XXTERN handle      acc_next_loconn PROTO_PARAMS((handle port, handle loconn));
XXTERN handle      acc_next_modpath PROTO_PARAMS((handle mod_p, handle path));
XXTERN handle      acc_next_net PROTO_PARAMS((handle mod_handle, handle net));
XXTERN handle      acc_next_output PROTO_PARAMS((handle path, handle pathout));
XXTERN handle      acc_next_parameter PROTO_PARAMS((handle module_p, handle param));
XXTERN handle      acc_next_port PROTO_PARAMS((handle ref_obj_p, handle port));
XXTERN handle      acc_next_portout PROTO_PARAMS((handle mod_p, handle port));
XXTERN handle      acc_next_primitive PROTO_PARAMS((handle mod_handle, handle prim));
XXTERN handle      acc_next_scope PROTO_PARAMS((handle ref_scope_p, handle scope));
XXTERN handle      acc_next_specparam PROTO_PARAMS((handle module_p, handle sparam));
XXTERN handle      acc_next_tchk PROTO_PARAMS((handle mod_p, handle tchk));
XXTERN handle      acc_next_terminal PROTO_PARAMS((handle gate_handle, handle term));
XXTERN handle      acc_next_topmod PROTO_PARAMS((handle topmod));
XXTERN PLI_INT32   acc_object_of_type PROTO_PARAMS((handle object, PLI_INT32 type));
XXTERN PLI_INT32   acc_object_in_typelist PROTO_PARAMS((handle object, PLI_INT32
*type_list));
XXTERN PLI_INT32   acc_product_type PROTO_PARAMS((void));
XXTERN PLI_BYTE8  *acc_product_version PROTO_PARAMS((void));
XXTERN PLI_INT32   acc_release_object PROTO_PARAMS((handle obj));
XXTERN PLI_INT32   acc_replace_delays PROTO_PARAMS((handle object, ...));
XXTERN PLI_INT32   acc_replace_pulsere PROTO_PARAMS((handle object, double val1r,
double val1x, ...));
XXTERN void        acc_reset_buffer PROTO_PARAMS((void));
XXTERN PLI_INT32   acc_set_interactive_scope PROTO_PARAMS((handle scope, PLI_INT32
callback_flag));
XXTERN PLI_INT32   acc_set_pulsere PROTO_PARAMS((handle path_p, double val1r, double
val1e));
XXTERN PLI_BYTE8  *acc_set_scope PROTO_PARAMS((handle object, ...));
XXTERN PLI_INT32   acc_set_value PROTO_PARAMS((handle obj, p_setval_value setval_p,
p_setval_delay delay_p));
XXTERN void        acc_vcl_add PROTO_PARAMS((handle object_p, PLI_INT32
(*consumer)(p_vc_record), PLI_BYTE8 *user_data, PLI_INT32 vcl_flags));
XXTERN void        acc_vcl_delete PROTO_PARAMS((handle object_p, PLI_INT32
(*consumer)(p_vc_record), PLI_BYTE8 *user_data, PLI_INT32 vcl_flags));
XXTERN PLI_BYTE8  *acc_version PROTO_PARAMS((void));


/*---------------------------------------------------------------------------*/
/*----------------------- global variable definitions -----------------------*/
/*---------------------------------------------------------------------------*/

PLI_VEXTERN PLI_DLLISPEC PLI_INT32 acc_error_flag;


/*---------------------------------------------------------------------------*/
/*--------------------------- macro definitions -----------------------------*/
/*---------------------------------------------------------------------------*/

#define  acc_handle_calling_mod_m  acc_handle_parent((handle)tf_getinstance())

#undef PLI_EXTERN
#undef PLI_VEXTERN

#ifdef ACC_USER_DEFINED_DLLISPEC
#undef ACC_USER_DEFINED_DLLISPEC
#undef PLI_DLLISPEC
#endif
#ifdef ACC_USER_DEFINED_DLLESPEC
#undef ACC_USER_DEFINED_DLLESPEC
#undef PLI_DLLESPEC
#endif
```

 Copyright 2000 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

```
#ifdef PLI_PROTOTYPES
#undef PLI_PROTOTYPES
#undef PROTO_PARAMS
#undef XXTERN
#undef EETERN
#endif

#endif /* ACC_USER_H */
```


Copyright 2000 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

 Copyright 2000 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

Annex F

veriuser.h

```
/************************************************************************
 * veriuser.h
 *
 * IEEE 1364-2000 Verilog HDL Programming Language Interface (PLI).
 *
 * This file contains the constant definitions, structure definitions, and
 * routine declarations for the Verilog Programming Language Interface TF
 * task/function routines.
 *
 ************************************************************************/

#ifndef VERIUSER_H
#define VERIUSER_H

/*-------------------------------------------------------------------------*/
/*---------------------------- Portability Help ---------------------------*/
/*-------------------------------------------------------------------------*/
/* Sized variables */
#ifndef PLI_TYPES
#define PLI_TYPES
typedef int             PLI_INT32;
typedef unsigned int    PLI_UINT32;
typedef short           PLI_INT16;
typedef unsigned short  PLI_UINT16;
typedef char            PLI_BYTE8;
typedef unsigned char   PLI_UBYTE8;
#endif

/* export a symbol */
#if WIN32
#ifndef PLI_DLLISPEC
#define PLI_DLLISPEC __declspec(dllimport)
#define VERIUSER_DEFINED_DLLISPEC 1
#endif
#else
#ifndef PLI_DLLISPEC
#define PLI_DLLISPEC
#endif
#endif

/* import a symbol */
#if WIN32
#ifndef PLI_DLLESPEC
#define PLI_DLLESPEC __declspec(dllexport)
#define VERIUSER_DEFINED_DLLESPEC 1
#endif
#else
#ifndef PLI_DLLESPEC
#define PLI_DLLESPEC
#endif
#endif

/* mark a function as external */
#ifndef PLI_EXTERN
```


Copyright 2000 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

```
#define PLI_EXTERN
#endif

/* mark a variable as external */
#ifndef PLI_VEXTERN
#define PLI_VEXTERN extern
#endif

#ifndef PLI_PROTOTYPES
#define PLI_PROTOTYPES
#define PROTO_PARAMS(params) params
/* object is defined imported by the application */
#define XXTERN PLI_EXTERN PLI_DLLISPEC
/* object is exported by the application */
#define EETERN PLI_EXTERN PLI_DLLESPEC
#endif

/*
 * The following group of defines exists purely for backwards compatibility
 */
#ifndef PLI_EXTRAS
#define PLI_EXTRAS
#define bool                    int
#define true                    1
#define TRUE                    1
#define false                   0
#define FALSE                   0
#define null                    0L
#endif


/*---------------------------------------------------------------------------*/
/*------------------------------ definitions --------------------------------*/
/*---------------------------------------------------------------------------*/

/*---------------------- defines for error interception ---------------------*/

#define ERR_MESSAGE             1
#define ERR_WARNING             2
#define ERR_ERROR               3
#define ERR_INTERNAL            4
#define ERR_SYSTEM              5

/*-------------- values for reason parameter to misctf routines -------------*/

#define reason_checktf          1
#define REASON_CHECKTF          reason_checktf
#define reason_sizetf           2
#define REASON_SIZETF           reason_sizetf
#define reason_calltf           3
#define REASON_CALLTF           reason_calltf
#define reason_save             4
#define REASON_SAVE             reason_save
#define reason_restart          5
#define REASON_RESTART          reason_restart
#define reason_disable          6
#define REASON_DISABLE          reason_disable
#define reason_paramvc          7
#define REASON_PARAMVC          reason_paramvc
#define reason_synch            8
#define REASON_SYNCH            reason_synch
#define reason_finish           9
#define REASON_FINISH           reason_finish
#define reason_reactivate      10
```

 Copyright 2000 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

```
#define REASON_REACTIVATE      reason_reactivate
#define reason_rosynch        11
#define REASON_ROSYNCH         reason_rosynch
#define reason_paramdrc       15
#define REASON_PARAMDRC        reason_paramdrc
#define reason_endofcompile   16
#define REASON_ENDOFCOMPILE    reason_endofcompile
#define reason_scope          17
#define REASON_SCOPE          reason_scope
#define reason_interactive    18
#define REASON_INTERACTIVE     reason_interactive
#define reason_reset          19
#define REASON_RESET           reason_reset
#define reason_endofreset     20
#define REASON_ENDOFRESET      reason_endofreset
#define reason_force          21
#define REASON_FORCE           reason_force
#define reason_release        22
#define REASON_RELEASE         reason_release
#define reason_startofsave    27
#define reason_startofrestart 28
#define REASON_MAX            28

/*-- types used by tf_typep() and expr_type field in tf_exprinfo structure --*/
#define tf_nullparam           0
#define TF_NULLPARAM           tf_nullparam
#define tf_string              1
#define TF_STRING              tf_string
#define tf_specialparam        2
#define TF_SPECIALPARAM        tf_specialparam
#define tf_readonly           10
#define TF_READONLY            tf_readonly
#define tf_readwrite          11
#define TF_READWRITE           tf_readwrite
#define tf_rwbitselect        12
#define TF_RWBITSELECT         tf_rwbitselect
#define tf_rwpartselect       13
#define TF_RWPARTSELECT        tf_rwpartselect
#define tf_rwmemselect        14
#define TF_RWMEMSELECT         tf_rwmemselect
#define tf_readonlyreal       15
#define TF_READONLYREAL        tf_readonlyreal
#define tf_readwritereal      16
#define TF_READWRITEREAL       tf_readwritereal

/*---------- types used by node_type field in tf_nodeinfo structure ---------*/
#define tf_null_node         100
#define TF_NULL_NODE           tf_null_node
#define tf_reg_node          101
#define TF_REG_NODE            tf_reg_node
#define tf_integer_node      102
#define TF_INTEGER_NODE        tf_integer_node
#define tf_time_node         103
#define TF_TIME_NODE           tf_time_node
#define tf_netvector_node    104
#define TF_NETVECTOR_NODE      tf_netvector_node
#define tf_netscalar_node    105
#define TF_NETSCALAR_NODE      tf_netscalar_node
#define tf_memory_node       106
#define TF_MEMORY_NODE         tf_memory_node
#define tf_real_node         107
#define TF_REAL_NODE           tf_real_node

/*--------------------------------------------------------------------------*/
```


Copyright 2000 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

```
/*-------------------------- structure definitions -------------------------*/
/*--------------------------------------------------------------------------*/

/*----- structure used with tf_exprinfo() to get expression information -----*/
typedef struct t_tfexprinfo
{
  PLI_INT16        expr_type;
  PLI_INT16        padding;
  struct t_vecval *expr_value_p;
  double           real_value;
  PLI_BYTE8       *expr_string;
  PLI_INT32        expr_ngroups;
  PLI_INT32        expr_vec_size;
  PLI_INT32        expr_sign;
  PLI_INT32        expr_lhs_select;
  PLI_INT32        expr_rhs_select;
} s_tfexprinfo, *p_tfexprinfo;

/*------- structure for use with tf_nodeinfo() to get node information ------*/
typedef struct t_tfnodeinfo
{
  PLI_INT16  node_type;
  PLI_INT16  padding;
  union
    {
      struct t_vecval      *vecval_p;
      struct t_strengthval *strengthval_p;
      PLI_BYTE8            *memoryval_p;
      double               *real_val_p;
    } node_value;
  PLI_BYTE8 *node_symbol;
  PLI_INT32  node_ngroups;
  PLI_INT32  node_vec_size;
  PLI_INT32  node_sign;
  PLI_INT32  node_ms_index;
  PLI_INT32  node_ls_index;
  PLI_INT32  node_mem_size;
  PLI_INT32  node_lhs_element;
  PLI_INT32  node_rhs_element;
  PLI_INT32 *node_handle;
} s_tfnodeinfo, *p_tfnodeinfo;

/*---------------------- data structure of vector values ---------------------*/
typedef struct t_vecval
{
  PLI_INT32 avalbits;
  PLI_INT32 bvalbits;
} s_vecval, *p_vecval;

/*--------------- data structure of scalar net strength values --------------*/
typedef struct t_strengthval
{
  PLI_INT32 strength0;
  PLI_INT32 strength1;
} s_strengthval, *p_strengthval;


/*--------------------------------------------------------------------------*/
/*--------------------------- routine definitions --------------------------*/
/*--------------------------------------------------------------------------*/

XXTERN void             io_mcdprintf PROTO_PARAMS((PLI_INT32 mcd, PLI_BYTE8 *format,
...));
XXTERN void             io_printf PROTO_PARAMS((PLI_BYTE8 *format, ...));
```


Copyright 2000 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

```
XXTERN PLI_BYTE8       *mc_scan_plusargs PROTO_PARAMS((PLI_BYTE8 *plusarg));
XXTERN PLI_INT32        tf_add_long PROTO_PARAMS((PLI_INT32 *aof_lowtime1, PLI_INT32
*aof_hightime1, PLI_INT32 lowtime2, PLI_INT32 hightime2));
XXTERN PLI_INT32        tf_asynchoff PROTO_PARAMS((void));
XXTERN PLI_INT32        tf_asynchon PROTO_PARAMS((void));
XXTERN PLI_INT32        tf_clearalldelays PROTO_PARAMS((void));
XXTERN PLI_INT32        tf_compare_long PROTO_PARAMS((PLI_UINT32 low1, PLI_UINT32
high1, PLI_UINT32 low2, PLI_UINT32 high2));
XXTERN PLI_INT32        tf_copypvc_flag PROTO_PARAMS((PLI_INT32 nparam));
XXTERN void             tf_divide_long PROTO_PARAMS((PLI_INT32 *aof_low1, PLI_INT32
*aof_high1, PLI_INT32 low2, PLI_INT32 high2));
XXTERN PLI_INT32        tf_dofinish PROTO_PARAMS((void));
XXTERN PLI_INT32        tf_dostop PROTO_PARAMS((void));
XXTERN PLI_INT32        tf_error PROTO_PARAMS((PLI_BYTE8 *fmt, ...));
XXTERN PLI_INT32        tf_evaluatep PROTO_PARAMS((PLI_INT32 pnum));
XXTERN p_tfexprinfo     tf_exprinfo PROTO_PARAMS((PLI_INT32 pnum, p_tfexprinfo pinfo));
XXTERN PLI_BYTE8       *tf_getcstringp PROTO_PARAMS((PLI_INT32 nparam));
XXTERN PLI_BYTE8       *tf_getinstance PROTO_PARAMS((void));
XXTERN PLI_INT32        tf_getlongp PROTO_PARAMS((PLI_INT32 *aof_highvalue, PLI_INT32
pnum));
XXTERN PLI_INT32        tf_getlongtime PROTO_PARAMS((PLI_INT32 *aof_hightime));
XXTERN PLI_INT32        tf_getnextlongtime PROTO_PARAMS((PLI_INT32 *aof_lowtime,
PLI_INT32 *aof_hightime));
XXTERN PLI_INT32        tf_getp PROTO_PARAMS((PLI_INT32 pnum));
XXTERN PLI_INT32        tf_getpchange PROTO_PARAMS((PLI_INT32 nparam));
XXTERN double           tf_getrealp PROTO_PARAMS((PLI_INT32 pnum));
XXTERN double           tf_getrealtime PROTO_PARAMS((void));
XXTERN PLI_BYTE8       *tf_gettflist PROTO_PARAMS((void));
XXTERN PLI_INT32        tf_gettime PROTO_PARAMS((void));
XXTERN PLI_INT32        tf_gettimeprecision PROTO_PARAMS((void));
XXTERN PLI_INT32        tf_gettimeunit PROTO_PARAMS((void));
XXTERN PLI_BYTE8       *tf_getworkarea PROTO_PARAMS((void));
XXTERN PLI_INT32        tf_iasynchoff PROTO_PARAMS((PLI_BYTE8 *inst));
XXTERN PLI_INT32        tf_iasynchon PROTO_PARAMS((PLI_BYTE8 *inst));
XXTERN PLI_INT32        tf_iclearalldelays PROTO_PARAMS((PLI_BYTE8 *inst));
XXTERN PLI_INT32        tf_icopypvc_flag PROTO_PARAMS((PLI_INT32 nparam, PLI_BYTE8
*inst));
XXTERN PLI_INT32        tf_ievaluatep PROTO_PARAMS((PLI_INT32 pnum, PLI_BYTE8 *inst));
XXTERN p_tfexprinfo     tf_iexprinfo PROTO_PARAMS((PLI_INT32 pnum, p_tfexprinfo pinfo,
PLI_BYTE8 *inst));
XXTERN PLI_BYTE8       *tf_igetcstringp PROTO_PARAMS((PLI_INT32 nparam, PLI_BYTE8
*inst));
XXTERN PLI_INT32        tf_igetlongp PROTO_PARAMS((PLI_INT32 *aof_highvalue, PLI_INT32
pnum, PLI_BYTE8 *inst));
XXTERN PLI_INT32        tf_igetlongtime PROTO_PARAMS((PLI_INT32 *aof_hightime,
PLI_BYTE8 *inst));
XXTERN PLI_INT32        tf_igetp PROTO_PARAMS((PLI_INT32 pnum, PLI_BYTE8 *inst));
XXTERN PLI_INT32        tf_igetpchange PROTO_PARAMS((PLI_INT32 nparam, PLI_BYTE8
*inst));
XXTERN double           tf_igetrealp PROTO_PARAMS((PLI_INT32 pnum, PLI_BYTE8 *inst));
XXTERN double           tf_igetrealtime PROTO_PARAMS((PLI_BYTE8 *inst));
XXTERN PLI_INT32        tf_igettime PROTO_PARAMS((PLI_BYTE8 *inst));
XXTERN PLI_INT32        tf_igettimeprecision PROTO_PARAMS((PLI_BYTE8 *inst));
XXTERN PLI_INT32        tf_igettimeunit PROTO_PARAMS((PLI_BYTE8 *inst));
XXTERN PLI_BYTE8       *tf_igetworkarea PROTO_PARAMS((PLI_BYTE8 *inst));
XXTERN PLI_BYTE8       *tf_imipname PROTO_PARAMS((PLI_BYTE8 *cell));
XXTERN PLI_INT32        tf_imovepvc_flag PROTO_PARAMS((PLI_INT32 nparam, PLI_BYTE8
*inst));
XXTERN p_tfnodeinfo     tf_inodeinfo PROTO_PARAMS((PLI_INT32 pnum, p_tfnodeinfo pinfo,
PLI_BYTE8 *inst));
XXTERN PLI_INT32        tf_inump PROTO_PARAMS((PLI_BYTE8 *inst));
XXTERN PLI_INT32        tf_ipropagatep PROTO_PARAMS((PLI_INT32 pnum, PLI_BYTE8 *inst));
XXTERN PLI_INT32        tf_iputlongp PROTO_PARAMS((PLI_INT32 pnum, PLI_INT32 lowvalue,
PLI_INT32 highvalue, PLI_BYTE8 *inst));
```


Copyright 2000 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

```
XXTERN PLI_INT32        tf_iputp PROTO_PARAMS((PLI_INT32 pnum, PLI_INT32 value,
PLI_BYTE8 *inst));
XXTERN PLI_INT32        tf_iputrealp PROTO_PARAMS((PLI_INT32 pnum, double value,
PLI_BYTE8 *inst));
XXTERN PLI_INT32        tf_irosynchronize PROTO_PARAMS((PLI_BYTE8 *inst));
XXTERN PLI_INT32        tf_isetdelay PROTO_PARAMS((PLI_INT32 delay, PLI_BYTE8 *inst));
XXTERN PLI_INT32        tf_isetlongdelay PROTO_PARAMS((PLI_INT32 lowdelay, PLI_INT32
highdelay, PLI_BYTE8 *inst));
XXTERN PLI_INT32        tf_isetrealdelay PROTO_PARAMS((double realdelay, PLI_BYTE8
*inst));
XXTERN PLI_INT32        tf_isetworkarea PROTO_PARAMS((PLI_BYTE8 *workarea, PLI_BYTE8
*inst));
XXTERN PLI_INT32        tf_isizep PROTO_PARAMS((PLI_INT32 pnum, PLI_BYTE8 *inst));
XXTERN PLI_BYTE8       *tf_ispname PROTO_PARAMS((PLI_BYTE8 *cell));
XXTERN PLI_INT32        tf_istrdelputp PROTO_PARAMS((PLI_INT32 nparam, PLI_INT32
bitlength, PLI_INT32 format_char, PLI_BYTE8 *value_p, PLI_INT32 delay, PLI_INT32
delaytype, PLI_BYTE8 *inst));
XXTERN PLI_BYTE8       *tf_istrgetp PROTO_PARAMS((PLI_INT32 pnum, PLI_INT32
format_char, PLI_BYTE8 *inst));
XXTERN PLI_INT32        tf_istrlongdelputp PROTO_PARAMS((PLI_INT32 nparam, PLI_INT32
bitlength, PLI_INT32 format_char, PLI_BYTE8 *value_p, PLI_INT32 lowdelay, PLI_INT32
highdelay, PLI_INT32 delaytype, PLI_BYTE8 *inst));
XXTERN PLI_INT32        tf_istrrealdelputp PROTO_PARAMS((PLI_INT32 nparam, PLI_INT32
bitlength, PLI_INT32 format_char, PLI_BYTE8 *value_p, double realdelay, PLI_INT32
delaytype, PLI_BYTE8 *inst));
XXTERN PLI_INT32        tf_isynchronize PROTO_PARAMS((PLI_BYTE8 *inst));
XXTERN PLI_INT32        tf_itestpvc_flag PROTO_PARAMS((PLI_INT32 nparam, PLI_BYTE8
*inst));
XXTERN PLI_INT32        tf_itypep PROTO_PARAMS((PLI_INT32 pnum, PLI_BYTE8 *inst));
XXTERN void             tf_long_to_real PROTO_PARAMS((PLI_INT32 int_lo, PLI_INT32
int_hi, double *aof_real));
XXTERN PLI_BYTE8       *tf_longtime_tostr PROTO_PARAMS((PLI_INT32 lowtime, PLI_INT32
hightime));
XXTERN PLI_INT32        tf_message PROTO_PARAMS((PLI_INT32 level, PLI_BYTE8 *facility,
PLI_BYTE8 *messno, PLI_BYTE8 *message, ...));
XXTERN PLI_BYTE8       *tf_mipname PROTO_PARAMS((void));
XXTERN PLI_INT32        tf_movepvc_flag PROTO_PARAMS((PLI_INT32 nparam));
XXTERN void             tf_multiply_long PROTO_PARAMS((PLI_INT32 *aof_low1, PLI_INT32
*aof_high1, PLI_INT32 low2, PLI_INT32 high2));
XXTERN p_tfnodeinfo     tf_nodeinfo PROTO_PARAMS((PLI_INT32 pnum, p_tfnodeinfo pinfo));
XXTERN PLI_INT32        tf_nump PROTO_PARAMS((void));
XXTERN PLI_INT32        tf_propagatep PROTO_PARAMS((PLI_INT32 pnum));
XXTERN PLI_INT32        tf_putlongp PROTO_PARAMS((PLI_INT32 pnum, PLI_INT32 lowvalue,
PLI_INT32 highvalue));
XXTERN PLI_INT32        tf_putp PROTO_PARAMS((PLI_INT32 pnum, PLI_INT32 value));
XXTERN PLI_INT32        tf_putrealp PROTO_PARAMS((PLI_INT32 pnum, double value));
XXTERN PLI_INT32        tf_read_restart PROTO_PARAMS((PLI_BYTE8 *blockptr, PLI_INT32
blocklen));
XXTERN void             tf_real_to_long PROTO_PARAMS((double real, PLI_INT32
*aof_int_lo, PLI_INT32 *aof_int_hi));
XXTERN PLI_INT32        tf_rosynchronize PROTO_PARAMS((void));
XXTERN void             tf_scale_longdelay PROTO_PARAMS((PLI_BYTE8 *cell, PLI_INT32
delay_lo, PLI_INT32 delay_hi, PLI_INT32 *aof_delay_lo, PLI_INT32 *aof_delay_hi));
XXTERN void             tf_scale_realdelay PROTO_PARAMS((PLI_BYTE8 *cell, double
realdelay, double *aof_realdelay));
XXTERN PLI_INT32        tf_setdelay PROTO_PARAMS((PLI_INT32 delay));
XXTERN PLI_INT32        tf_setlongdelay PROTO_PARAMS((PLI_INT32 lowdelay, PLI_INT32
highdelay));
XXTERN PLI_INT32        tf_setrealdelay PROTO_PARAMS((double realdelay));
XXTERN PLI_INT32        tf_setworkarea PROTO_PARAMS((PLI_BYTE8 *workarea));
XXTERN PLI_INT32        tf_sizep PROTO_PARAMS((PLI_INT32 pnum));
XXTERN PLI_BYTE8       *tf_spname PROTO_PARAMS((void));
```


Copyright 2000 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

```
XXTERN PLI_INT32        tf_strdelputp PROTO_PARAMS((PLI_INT32 nparam, PLI_INT32
bitlength, PLI_INT32 format_char, PLI_BYTE8 *value_p, PLI_INT32 delay, PLI_INT32
delaytype));
XXTERN PLI_BYTE8       *tf_strgetp PROTO_PARAMS((PLI_INT32 pnum, PLI_INT32
format_char));
XXTERN PLI_BYTE8       *tf_strgettime PROTO_PARAMS((void));
XXTERN PLI_INT32        tf_strlongdelputp PROTO_PARAMS((PLI_INT32 nparam, PLI_INT32
bitlength, PLI_INT32 format_char, PLI_BYTE8 *value_p, PLI_INT32 lowdelay, PLI_INT32
highdelay, PLI_INT32 delaytype));
XXTERN PLI_INT32        tf_strrealdelputp PROTO_PARAMS((PLI_INT32 nparam, PLI_INT32
bitlength, PLI_INT32 format_char, PLI_BYTE8 *value_p, double realdelay, PLI_INT32
delaytype));
XXTERN PLI_INT32        tf_subtract_long PROTO_PARAMS((PLI_INT32 *aof_lowtime1,
PLI_INT32 *aof_hightime1, PLI_INT32 lowtime2, PLI_INT32 hightime2));
XXTERN PLI_INT32        tf_synchronize PROTO_PARAMS((void));
XXTERN PLI_INT32        tf_testpvc_flag PROTO_PARAMS((PLI_INT32 nparam));
XXTERN PLI_INT32        tf_text PROTO_PARAMS((PLI_BYTE8 *fmt, ...));
XXTERN PLI_INT32        tf_typep PROTO_PARAMS((PLI_INT32 pnum));
XXTERN void             tf_unscale_longdelay PROTO_PARAMS((PLI_BYTE8 *cell, PLI_INT32
delay_lo, PLI_INT32 delay_hi, PLI_INT32 *aof_delay_lo, PLI_INT32 *aof_delay_hi));
XXTERN void             tf_unscale_realdelay PROTO_PARAMS((PLI_BYTE8 *cell, double
realdelay, double *aof_realdelay));
XXTERN PLI_INT32        tf_warning PROTO_PARAMS((PLI_BYTE8 *fmt, ...));
XXTERN PLI_INT32        tf_write_save PROTO_PARAMS((PLI_BYTE8 *blockptr, PLI_INT32
blocklen));
XXTERN PLI_BYTE8       *tf_getroutine PROTO_PARAMS((void));
XXTERN PLI_BYTE8       *tf_igetroutine PROTO_PARAMS((PLI_BYTE8 *inst));

EETERN PLI_INT32        err_intercept PROTO_PARAMS((PLI_INT32 level, PLI_BYTE8
*facility, PLI_BYTE8 *code));

/*-------------------------------------------------------------------------*/
/*---------------------------------- Globals ------------------------------*/
/*-------------------------------------------------------------------------*/

PLI_VEXTERN PLI_DLLESPEC PLI_BYTE8 *veriuser_version_str;
PLI_VEXTERN PLI_DLLESPEC PLI_INT32 (*endofcompile_routines[])();

#undef PLI_EXTERN
#undef PLI_VEXTERN

#ifdef VERIUSER_DEFINED_DLLISPEC
#undef VERIUSER_DEFINED_DLLISPEC
#undef PLI_DLLISPEC
#endif
#ifdef VERIUSER_DEFINED_DLLESPEC
#undef VERIUSER_DEFINED_DLLESPEC
#undef PLI_DLLESPEC
#endif

#ifdef PLI_PROTOTYPES
#undef PLI_PROTOTYPES
#undef PROTO_PARAMS
#undef XXTERN
#undef EETERN
#endif

#endif /* VERIUSER_H */
```


Copyright 2000 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

 Copyright 2000 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

Annex G

vpi_user.h

```
/*************************************************************************
 * vpi_user.h
 *
 * IEEE 1364-2000 Verilog HDL Programming Language Interface (PLI).
 *
 * This file contains the constant definitions, structure definitions, and
 * routine declarations used by the Verilog PLI procedural interface VPI
 * access routines.
 *
 * The file should be included with all C routines that use the PLI VPI
 * routines.
 *************************************************************************/

/*************************************************************************
 * NOTE: the constant values 1 through 299 are reserved for use in this
 * vpi_user.h file.
 *************************************************************************/

#ifndef VPI_USER_H
#define VPI_USER_H

#include <stdarg.h>

/*-------------------------------------------------------------------------*/
/*-------------------------- Portability Help -----------------------------*/
/*-------------------------------------------------------------------------*/

/* Sized variables */
#ifndef PLI_TYPES
#define PLI_TYPES
typedef int             PLI_INT32;
typedef unsigned int    PLI_UINT32;
typedef short           PLI_INT16;
typedef unsigned short  PLI_UINT16;
typedef char            PLI_BYTE8;
typedef unsigned char   PLI_UBYTE8;
#endif

/* Use to export a symbol */
#if WIN32
#ifndef PLI_DLLISPEC
#define PLI_DLLISPEC __declspec(dllimport)
#define VPI_USER_DEFINED_DLLISPEC 1
#endif
#else
#ifndef PLI_DLLISPEC
#define PLI_DLLISPEC
#endif
#endif

/* Use to import a symbol */
#if WIN32
#ifndef PLI_DLLESPEC
#define PLI_DLLESPEC __declspec(dllexport)
```


Copyright 2000 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

```
#define VPI_USER_DEFINED_DLLESPEC 1
#endif
#else
#ifndef PLI_DLLESPEC
#define PLI_DLLESPEC
#endif
#endif

/* Use to mark a function as external */
#ifndef PLI_EXTERN
#define PLI_EXTERN
#endif

/* Use to mark a variable as external */
#ifndef PLI_VEXTERN
#define PLI_VEXTERN extern
#endif

#ifndef PLI_PROTOTYPES
#define PLI_PROTOTYPES
#define PROTO_PARAMS(params) params
/* object is defined imported by the application */
#define XXTERN PLI_EXTERN PLI_DLLISPEC
/* object is exported by the application */
#define EETERN PLI_EXTERN PLI_DLLESPEC
#endif


/******************************** TYPEDEFS *******************************/

typedef PLI_UINT32 *vpiHandle;

/***************************** OBJECT TYPES ******************************/
#define vpiAlways             1   /* always block */
#define vpiAssignStmt         2   /* quasi-continuous assignment */
#define vpiAssignment         3   /* procedural assignment */
#define vpiBegin              4   /* block statement */
#define vpiCase               5   /* case statement */
#define vpiCaseItem           6   /* case statement item */
#define vpiConstant           7   /* numerical constant or literal string */
#define vpiContAssign         8   /* continuous assignment */
#define vpiDeassign           9   /* deassignment statement */
#define vpiDefParam          10   /* defparam */
#define vpiDelayControl      11   /* delay statement (e.g. #10) */
#define vpiDisable           12   /* named block disable statement */
#define vpiEventControl      13   /* wait on event, e.g. @e */
#define vpiEventStmt         14   /* event trigger, e.g. ->e */
#define vpiFor               15   /* for statement */
#define vpiForce             16   /* force statement */
#define vpiForever           17   /* forever statement */
#define vpiFork              18   /* fork-join block */
#define vpiFuncCall          19   /* HDL function call */
#define vpiFunction          20   /* HDL function */
#define vpiGate              21   /* primitive gate */
#define vpiIf                22   /* if statement */
#define vpiIfElse            23   /* if-else statement */
#define vpiInitial           24   /* initial block */
#define vpiIntegerVar        25   /* integer variable */
#define vpiInterModPath      26   /* intermodule wire delay */
#define vpiIterator          27   /* iterator */
#define vpiIODecl            28   /* input/output declaration */
#define vpiMemory            29   /* behavioral memory */
#define vpiMemoryWord        30   /* single word of memory */
#define vpiModPath           31   /* module path for path delays */
```

 Copyright 2000 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

```
#define vpiModule            32   /* module instance */
#define vpiNamedBegin        33   /* named block statement */
#define vpiNamedEvent        34   /* event variable */
#define vpiNamedFork         35   /* named fork-join block */
#define vpiNet               36   /* scalar or vector net */
#define vpiNetBit            37   /* bit of vector net */
#define vpiNullStmt          38   /* a semicolon. Ie. #10 ; */
#define vpiOperation         39   /* behavioral operation */
#define vpiParamAssign       40   /* module parameter assignment */
#define vpiParameter         41   /* module parameter */
#define vpiPartSelect        42   /* part select */
#define vpiPathTerm          43   /* terminal of module path */
#define vpiPort              44   /* module port */
#define vpiPortBit           45   /* bit of vector module port */
#define vpiPrimTerm          46   /* primitive terminal */
#define vpiRealVar           47   /* real variable */
#define vpiReg               48   /* scalar or vector register */
#define vpiRegBit            49   /* bit of vector register net */
#define vpiRelease           50   /* release statement */
#define vpiRepeat            51   /* repeat statement */
#define vpiRepeatControl     52   /* repeat control in an assign stmt */
#define vpiSchedEvent        53   /* vpi_put_value() event */
#define vpiSpecParam         54   /* specparam */
#define vpiSwitch            55   /* transistor switch */
#define vpiSysFuncCall       56   /* system function call */
#define vpiSysTaskCall       57   /* system task call */
#define vpiTableEntry        58   /* UDP state table entry */
#define vpiTask              59   /* HDL task */
#define vpiTaskCall          60   /* HDL task call */
#define vpiTchk              61   /* timing check */
#define vpiTchkTerm          62   /* terminal of timing check */
#define vpiTimeVar           63   /* time variable */
#define vpiTimeQueue         64   /* simulation event queue */
#define vpiUdp               65   /* user-defined primitive */
#define vpiUdpDefn           66   /* UDP definition */
#define vpiUserSystf         67   /* user defined system task or function */
#define vpiVarSelect         68   /* variable array selection */
#define vpiWait              69   /* wait statement */
#define vpiWhile             70   /* while statement */

/****************** object types added with 1364-2000 *******************/
#define vpiAttribute        105   /* attribute of an object */
#define vpiBitSelect        106   /* Bit select of parameter, var select */
#define vpiCallback         107   /* callback object */
#define vpiDelayTerm        108   /* Delay term which is a load or driver */
#define vpiDelayDevice      109   /* Delay object within a net */
#define vpiFrame            110   /* reentrant task/func frame */
#define vpiGateArray        111   /* gate instance array */
#define vpiModuleArray      112   /* module instance array */
#define vpiPrimitiveArray   113   /* vpiprimitiveArray type */
#define vpiNetWord          114   /* word of a multidimensional wire */
#define vpiRange            115   /* range declaration */
#define vpiRegWord          116   /* word of a multidimensional register */
#define vpiSwitchArray      117   /* switch instance array */
#define vpiUdpArray         118   /* UDP instance array */


/******************************** METHODS ********************************/
/************** methods used to traverse 1 to 1 relationships **************/
#define vpiCondition         71   /* condition expression */
#define vpiDelay             72   /* net or gate delay */
#define vpiElseStmt          73   /* else statement */
#define vpiForIncStmt        74   /* increment statement in for loop */
#define vpiForInitStmt       75   /* initialization statement in for loop */
```


Copyright 2000 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

```
#define vpiHighConn          76   /* higher connection to port */
#define vpiLhs               77   /* left-hand side of assignment */
#define vpiIndex             78   /* index of var select, bit select, etc. */
#define vpiLeftRange         79   /* left range of vector or part select */
#define vpiLowConn           80   /* lower connection to port */
#define vpiParent            81   /* parent object */
#define vpiRhs               82   /* right-hand side of assignment */
#define vpiRightRange        83   /* right range of vector or part select */
#define vpiScope             84   /* containing scope object */
#define vpiSysTfCall         85   /* task function call */
#define vpiTchkDataTerm      86   /* timing check data term */
#define vpiTchkNotifier      87   /* timing check notifier */
#define vpiTchkRefTerm       88   /* timing check reference term */

/************ methods used to traverse 1 to many relationships ************/
#define vpiArgument          89   /* argument to (system) task/function */
#define vpiBit               90   /* bit of vector net or port */
#define vpiDriver            91   /* driver for a net */
#define vpiInternalScope     92   /* internal scope in module */
#define vpiLoad              93   /* load on net or register */
#define vpiModDataPathIn     94   /* data terminal of a module path */
#define vpiModPathIn         95   /* Input terminal of a module path */
#define vpiModPathOut        96   /* output terminal of a module path */
#define vpiOperand           97   /* operand of expression */
#define vpiPortInst          98   /* connected port instance */
#define vpiProcess           99   /* process in module */
#define vpiVariables        100   /* variables in module */
#define vpiUse              101   /* usage */

/***** methods which can traverse 1 to 1, or 1 to many relationships ******/
#define vpiExpr             102   /* connected expression */
#define vpiPrimitive        103   /* primitive (gate, switch, UDP) */
#define vpiStmt             104   /* statement in process or task */

/******************** methods added with 1364-2000 **********************/
#define vpiActiveTimeFormat 119   /* active $timeformat() system task */
#define vpiInTerm           120   /* To get to a delay device's drivers. */
#define vpiInstanceArray    121   /* vpiInstance arrays */
#define vpiLocalDriver      122   /* local drivers (within a module */
#define vpiLocalLoad        123   /* local loads (within a module */
#define vpiOutTerm          124   /* To get to a delay device's loads. */
#define vpiPorts            125   /* Module port */
#define vpiSimNet           126   /* simulated net after collapsing */
#define vpiTaskFunc         127   /* HDL task or function */


/****************************** PROPERTIES ******************************/
/*********************** generic object properties **********************/
#define vpiUndefined        -1   /* undefined property */
#define vpiType              1   /* type of object */
#define vpiName              2   /* local name of object */
#define vpiFullName          3   /* full hierarchical name */
#define vpiSize              4   /* size of gate, net, port, etc. */
#define vpiFile              5   /* File name in which the object is used */
#define vpiLineNo            6   /* line number where the object is used */

/************************** module properties ***************************/
#define vpiTopModule         7   /* top-level module (boolean) */
#define vpiCellInstance      8   /* cell (boolean) */
#define vpiDefName           9   /* module definition name */
#define vpiProtected        10   /* source protected module (boolean) */
#define vpiTimeUnit         11   /* module time unit */
#define vpiTimePrecision    12   /* module time precision */
#define vpiDefNetType       13   /* default net type */
```


Copyright 2000 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

```
#define vpiUnconnDrive      14   /* unconnected port drive strength */
#define vpiHighZ              1   /* No default drive given */
#define vpiPull1              2   /* default pull1 drive */
#define vpiPull0              3   /* default pull0 drive */
#define vpiDefFile          15   /* File name where the module is defined */
#define vpiDefLineNo        16   /* line number for module definition */
#define vpiDefDelayMode     47   /* Default delay mode for a module */
#define vpiDelayModeNone      1
#define vpiDelayModePath      2
#define vpiDelayModeDistrib   3
#define vpiDelayModeUnit      4
#define vpiDelayModeZero      5
#define vpiDefDecayTime     48   /* Default decay time for a module */

/************************ port and net properties ************************/
#define vpiScalar           17   /* scalar (boolean) */
#define vpiVector           18   /* vector (boolean) */
#define vpiExplicitName     19   /* port is explicitly named */
#define vpiDirection        20   /* direction of port: */
#define vpiInput              1   /* input */
#define vpiOutput             2   /* output */
#define vpiInout              3   /* inout */
#define vpiMixedIO            4   /* mixed input-output */
#define vpiNoDirection        5   /* no direction */
#define vpiConnByName       21   /* connected by name (boolean) */

#define vpiNetType          22   /* net subtypes: */
#define vpiWire               1   /* wire net */
#define vpiWand               2   /* wire-and net */
#define vpiWor                3   /* wire-or net */
#define vpiTri                4   /* tri-state net */
#define vpiTri0               5   /* pull-down net */
#define vpiTri1               6   /* pull-up net */
#define vpiTriReg             7   /* tri state reg net */
#define vpiTriAnd             8   /* tri-state wire-and net */
#define vpiTriOr              9   /* tri-state wire-or net */
#define vpiSupply1           10   /* supply 1 net */
#define vpiSupply0           11   /* supply zero net */
#define vpiNone              12   /* no default net type (1364-2000) */

#define vpiExplicitScalared 23   /* explicitly scalared (boolean) */
#define vpiExplicitVectored 24   /* explicitly vectored (boolean) */
#define vpiExpanded         25   /* expanded vector net (boolean) */
#define vpiImplicitDecl     26   /* implicitly declared net (boolean) */
#define vpiChargeStrength   27   /* charge decay strength of net */
/*  Defined as part of strengths section.
#define vpiLargeCharge        0x10
#define vpiMediumCharge       0x04
#define vpiSmallCharge        0x02
*/
#define vpiArray            28   /* variable array (boolean) */
#define vpiPortIndex        29   /* Port index */

/********************** gate and terminal properties **********************/
#define vpiTermIndex        30   /* Index of a primitive terminal */
#define vpiStrength0        31   /* 0-strength of net or gate */
#define vpiStrength1        32   /* 1-strength of net or gate */
#define vpiPrimType         33   /* prmitive subtypes: */
#define vpiAndPrim            1   /* and gate */
#define vpiNandPrim           2   /* nand gate */
#define vpiNorPrim            3   /* nor gate */
#define vpiOrPrim             4   /* or gate */
#define vpiXorPrim            5   /* xor gate */
#define vpiXnorPrim           6   /* xnor gate */
```


Copyright 2000 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

```
#define vpiBufPrim           7   /* buffer */
#define vpiNotPrim           8   /* not gate */
#define vpiBufif0Prim        9   /* zero-enabled buffer */
#define vpiBufif1Prim       10   /* one-enabled buffer */
#define vpiNotif0Prim       11   /* zero-enabled not gate */
#define vpiNotif1Prim       12   /* one-enabled not gate */
#define vpiNmosPrim         13   /* nmos switch */
#define vpiPmosPrim         14   /* pmos switch */
#define vpiCmosPrim         15   /* cmos switch */
#define vpiRnmosPrim        16   /* resistive nmos switch */
#define vpiRpmosPrim        17   /* resistive pmos switch */
#define vpiRcmosPrim        18   /* resistive cmos switch */
#define vpiRtranPrim        19   /* resistive bidirectional */
#define vpiRtranif0Prim     20   /* zero-enable resistive bidirectional */
#define vpiRtranif1Prim     21   /* one-enable resistive bidirectional */
#define vpiTranPrim         22   /* bidirectional */
#define vpiTranif0Prim      23   /* zero-enabled bidirectional */
#define vpiTranif1Prim      24   /* one-enabled bidirectional */
#define vpiPullupPrim       25   /* pullup */
#define vpiPulldownPrim     26   /* pulldown */
#define vpiSeqPrim          27   /* sequential UDP */
#define vpiCombPrim         28   /* combinational UDP */

/************** path, path terminal, timing check properties **************/
#define vpiPolarity         34   /* polarity of module path... */
#define vpiDataPolarity     35   /* ...or data path: */
#define vpiPositive          1   /* positive */
#define vpiNegative          2   /* negative */
#define vpiUnknown           3   /* unknown (unspecified) */

#define vpiEdge             36   /* edge type of module path: */
#define vpiNoEdge   0x00000000     /* no edge */
#define vpiEdge01   0x00000001     /* 0 -> 1 */
#define vpiEdge10   0x00000002     /* 1 -> 0 */
#define vpiEdge0x   0x00000004     /* 0 -> x */
#define vpiEdgex1   0x00000008     /* x -> 1 */
#define vpiEdge1x   0x00000010     /* 1 -> x */
#define vpiEdgex0   0x00000020     /* x -> 0 */
#define vpiPosedge (vpiEdgex1 | vpiEdge01 | vpiEdge0x)
#define vpiNegedge (vpiEdgex0 | vpiEdge10 | vpiEdge1x)
#define vpiAnyEdge (vpiPosedge | vpiNegedge)

#define vpiPathType         37   /* path delay connection subtypes: */
#define vpiPathFull          1   /* ( a *> b ) */
#define vpiPathParallel      2   /* ( a => b ) */

#define vpiTchkType         38   /* timing check subtypes: */
#define vpiSetup             1   /* $setup */
#define vpiHold              2   /* $hold */
#define vpiPeriod            3   /* $period */
#define vpiWidth             4   /* $width */
#define vpiSkew              5   /* $skew */
#define vpiRecovery          6   /* $recovery */
#define vpiNoChange          7   /* $nochange */
#define vpiSetupHold         8   /* $setuphold */
#define vpiFullskew          9   /* $fullskew -- added for 1364-2000 */
#define vpiRecrem           10   /* $recrem   -- added for 1364-2000 */
#define vpiRemoval          11   /* $removal  -- added for 1364-2000 */
#define vpiTimeskew         12   /* $timeskew -- added for 1364-2000 */

/************************ expression properties ************************/
#define vpiOpType           39   /* operation subtypes: */
#define vpiMinusOp           1   /* unary minus */
#define vpiPlusOp            2   /* unary plus */
```

 Copyright 2000 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

```
#define vpiNotOp             3   /* unary not */
#define vpiBitNegOp          4   /* bitwise negation */
#define vpiUnaryAndOp        5   /* bitwise reduction and */
#define vpiUnaryNandOp       6   /* bitwise reduction nand */
#define vpiUnaryOrOp         7   /* bitwise reduction or */
#define vpiUnaryNorOp        8   /* bitwise reduction nor */
#define vpiUnaryXorOp        9   /* bitwise reduction xor */
#define vpiUnaryXNorOp      10   /* bitwise reduction xnor */
#define vpiSubOp            11   /* binary subtraction */
#define vpiDivOp            12   /* binary division */
#define vpiModOp            13   /* binary modulus */
#define vpiEqOp             14   /* binary equality */
#define vpiNeqOp            15   /* binary inequality */
#define vpiCaseEqOp         16   /* case (x and z) equality */
#define vpiCaseNeqOp        17   /* case inequality */
#define vpiGtOp             18   /* binary greater than */
#define vpiGeOp             19   /* binary greater than or equal */
#define vpiLtOp             20   /* binary less than */
#define vpiLeOp             21   /* binary less than or equal */
#define vpiLShiftOp         22   /* binary left shift */
#define vpiRShiftOp         23   /* binary right shift */
#define vpiAddOp            24   /* binary addition */
#define vpiMultOp           25   /* binary multiplication */
#define vpiLogAndOp         26   /* binary logical and */
#define vpiLogOrOp          27   /* binary logical or */
#define vpiBitAndOp         28   /* binary bitwise and */
#define vpiBitOrOp          29   /* binary bitwise or */
#define vpiBitXorOp         30   /* binary bitwise xor */
#define vpiBitXNorOp        31   /* binary bitwise xnor */
#define vpiBitXnorOp        vpiBitXNorOp /* added with 1364-2000 */
#define vpiConditionOp      32   /* ternary conditional */
#define vpiConcatOp         33   /* n-ary concatenation */
#define vpiMultiConcatOp    34   /* repeated concatenation */
#define vpiEventOrOp        35   /* event or */
#define vpiNullOp           36   /* null operation */
#define vpiListOp           37   /* list of expressions */
#define vpiMinTypMaxOp      38   /* min:typ:max: delay expression */
#define vpiPosedgeOp        39   /* posedge */
#define vpiNegedgeOp        40   /* negedge */
#define vpiArithLShiftOp    41   /* arithmetic left shift (1364-2000) */
#define vpiArithRShiftOp    42   /* arithmetic right shift (1364-2000) */
#define vpiPowerOp          43   /* arithmetic power op (1364-2000) */

#define vpiConstType       40   /* constant subtypes: */
#define vpiDecConst         1   /* decimal integer */
#define vpiRealConst        2   /* real */
#define vpiBinaryConst      3   /* binary integer */
#define vpiOctConst         4   /* octal integer */
#define vpiHexConst         5   /* hexadecimal integer */
#define vpiStringConst      6   /* string literal */
#define vpiIntConst         7   /* HDL integer constant (1364-2000) */

#define vpiBlocking        41   /* blocking assignment (boolean) */
#define vpiCaseType        42   /* case statement subtypes: */
#define vpiCaseExact        1   /* exact match */
#define vpiCaseX            2   /* ignore X's */
#define vpiCaseZ            3   /* ignore Z's */
#define vpiNetDeclAssign   43   /* assign part of decl (boolean) */

/************** task/function properties **************/
#define vpiFuncType        44   /* HDL function and system function type */
#define vpiIntFunc          1   /* returns integer */
#define vpiRealFunc         2   /* returns real */
#define vpiTimeFunc         3   /* returns time */
```


Copyright 2000 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

```
#define vpiSizedFunc          4   /* returns an arbitrary size */
#define vpiSizedSignedFunc    5   /* returns sized signed value */
/* alias 1364-1995 system function subtypes to 1364-2000 function subtypes */
#define vpiSysFuncType        vpiFuncType
#define vpiSysFuncInt         vpiIntFunc
#define vpiSysFuncReal        vpiRealFunc
#define vpiSysFuncTime        vpiTimeFunc
#define vpiSysFuncSized       vpiSizedFunc

#define vpiUserDefn         45   /* user defined system task/func (boolean) */
#define vpiScheduled        46   /* object still scheduled (boolean) */

/********************** properties added with 1364-2000 ******************/
#define vpiActive              49  /* reentrant task/func frame is active */
#define vpiAutomatic           50  /* task/func obj is automatic */
#define vpiCell                51  /* configuration cell */
#define vpiConfig              52  /* configuration config file */
#define vpiConstantSelect      53  /* (boolean) bit or part select indices
                                      are constant expressions */
#define vpiDecompile           54  /* decompile the object */
#define vpiDefAttribute        55  /* Attribute defined for the obj */
#define vpiDelayType           56  /* delay subtype */
#define vpiModPathDelay           1  /* module path delay */
#define vpiInterModPathDelay      2  /* intermodule path delay */
#define vpiMIPDelay               3  /* module input port delay */
#define vpiIterationType       57  /* object type of an iterator */
#define vpiLibrary             58  /* configuration library */
#define vpiMultiArray          59  /* Object is a multidimensional array */
#define vpiOffset              60  /* offset from LSB */
#define vpiResolvedNetType     61  /* net subtype after resolution, returns
                                      same subtypes as vpiNetType */
#define vpiSaveRestartID       62  /* unique ID for save/restart data */
#define vpiSaveRestartLocation 63  /* name of save/restart data file */
#define vpiValid               64  /* reentrant task/func frame is valid */
#define vpiSigned              65  /* TRUE when object's value is signed */
#define vpiLocalParam         .70  /* TRUE when a parameter is declared as localparam *
/*********** vpi_control() constants (added with 1364-2000) ***********/
#define vpiStop                66  /* execute simulator's $stop */
#define vpiFinish              67  /* execute simulator's $finish */
#define vpiReset               68  /* execute simulator's $reset */
#define vpiSetInteractiveScope 69  /* set simulator's interactive scope */

/************************* I/O related defines ***************************/
#define VPI_MCD_STDOUT  0x00000001

/************************* STRUCTURE DEFINITIONS ************************/

/***************************** time structure ****************************/
typedef struct t_vpi_time
{
  PLI_INT32  type;              /* [vpiScaledRealTime, vpiSimTime,
                                    vpiSuppressTime] */
  PLI_UINT32 high, low;         /* for vpiSimTime */
  double     real;              /* for vpiScaledRealTime */
} s_vpi_time, *p_vpi_time;

/* time types */
#define vpiScaledRealTime 1
#define vpiSimTime        2
#define vpiSuppressTime   3

/***************************** delay structures **************************/
typedef struct t_vpi_delay
{
```


Copyright 2000 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

```
  struct t_vpi_time *da;         /* pointer to user allocated array of
                                    delay values */
  PLI_INT32 no_of_delays;        /* number of delays */
  PLI_INT32 time_type;           /* [vpiScaledRealTime, vpiSimTime,
                                     vpiSuppressTime] */
  PLI_INT32 mtm_flag;            /* true for mtm values */
  PLI_INT32 append_flag;         /* true for append */
  PLI_INT32 pulsere_flag;        /* true for pulsere values */
} s_vpi_delay, *p_vpi_delay;

/*************************** value structures ***************************/
/* vector value */
typedef struct t_vpi_vecval
{
  /* following fields are repeated enough times to contain vector */
  PLI_INT32 aval, bval;          /* bit encoding: ab: 00=0, 10=1, 11=X, 01=Z */
} s_vpi_vecval, *p_vpi_vecval;

/* strength (scalar) value */
typedef struct t_vpi_strengthval
{
  PLI_INT32 logic;               /* vpi[0,1,X,Z] */
  PLI_INT32 s0, s1;              /* refer to strength coding below */
} s_vpi_strengthval, *p_vpi_strengthval;

/* strength values */
#define vpiSupplyDrive     0x80
#define vpiStrongDrive     0x40
#define vpiPullDrive       0x20
#define vpiWeakDrive       0x08
#define vpiLargeCharge     0x10
#define vpiMediumCharge    0x04
#define vpiSmallCharge     0x02
#define vpiHiZ             0x01

/* generic value */
typedef struct t_vpi_value
{
  PLI_INT32 format; /* vpi[[Bin,Oct,Dec,Hex]Str,Scalar,Int,Real,String,
                       Vector,Strength,Suppress,Time,ObjType]Val */
  union
    {
      PLI_BYTE8                *str;       /* string value */
      PLI_INT32                 scalar;    /* vpi[0,1,X,Z] */
      PLI_INT32                 integer;   /* integer value */
      double                    real;      /* real value */
      struct t_vpi_time        *time;      /* time value */
      struct t_vpi_vecval      *vector;    /* vector value */
      struct t_vpi_strengthval *strength;  /* strength value */
      PLI_BYTE8                *misc;      /* ...other */
    } value;
} s_vpi_value, *p_vpi_value;

/* value formats */
#define vpiBinStrVal          1
#define vpiOctStrVal          2
#define vpiDecStrVal          3
#define vpiHexStrVal          4
#define vpiScalarVal          5
#define vpiIntVal             6
#define vpiRealVal            7
#define vpiStringVal          8
#define vpiVectorVal          9
#define vpiStrengthVal       10
```


Copyright 2000 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

```
#define vpiTimeVal           11
#define vpiObjTypeVal        12
#define vpiSuppressVal       13

/* delay modes */
#define vpiNoDelay            1
#define vpiInertialDelay      2
#define vpiTransportDelay     3
#define vpiPureTransportDelay 4

/* force and release flags */
#define vpiForceFlag          5
#define vpiReleaseFlag        6

/* scheduled event cancel flag */
#define vpiCancelEvent        7

/* bit mask for the flags argument to vpi_put_value() */
#define vpiReturnEvent        0x1000

/* scalar values */
#define vpi0                  0
#define vpi1                  1
#define vpiZ                  2
#define vpiX                  3
#define vpiH                  4
#define vpiL                  5
#define vpiDontCare           6
/*
#define vpiNoChange           7   Defined under vpiTchkType, but
                                  can be used here.
*/

/******************** system task/function structure ********************/
typedef struct t_vpi_systf_data
{
  PLI_INT32 type;                     /* vpiSysTask, vpiSysFunc */
  PLI_INT32 sysfunctype;              /* vpiSysTask, vpi[Int,Real,Time,Sized,
                                                     SizedSigned]Func */
  PLI_BYTE8 *tfname;                  /* first character must be `$' */
  PLI_INT32 (*calltf)(PLI_BYTE8 *);
  PLI_INT32 (*compiletf)(PLI_BYTE8 *);
  PLI_INT32 (*sizetf)(PLI_BYTE8 *);   /* for sized function
                                         callbacks only */
  PLI_BYTE8 *user_data;
} s_vpi_systf_data, *p_vpi_systf_data;

#define vpiSysTask            1
#define vpiSysFunc            2
/* the subtypes are defined under the vpiFuncType property */

/***************** Verilog execution information structure ***************/
typedef struct t_vpi_vlog_info
{
  PLI_INT32 argc;
  PLI_BYTE8 **argv;
  PLI_BYTE8 *product;
  PLI_BYTE8 *version;
} s_vpi_vlog_info, *p_vpi_vlog_info;

/******************** PLI error information structure ********************/
typedef struct t_vpi_error_info
{
  PLI_INT32 state;          /* vpi[Compile,PLI,Run] */
```


Copyright 2000 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

```
  PLI_INT32 level;          /* vpi[Notice,Warning,Error,System,Internal] */
  PLI_BYTE8 *message;
  PLI_BYTE8 *product;
  PLI_BYTE8 *code;
  PLI_BYTE8 *file;
  PLI_INT32 line;
} s_vpi_error_info, *p_vpi_error_info;

/* error types */
#define vpiCompile             1
#define vpiPLI                 2
#define vpiRun                 3

#define vpiNotice              1
#define vpiWarning             2
#define vpiError               3
#define vpiSystem              4
#define vpiInternal            5


/************************* callback structures ***************************/
/* normal callback structure */
typedef struct t_cb_data
{
  PLI_INT32    reason;                       /* callback reason */
  PLI_INT32    (*cb_rtn)(struct t_cb_data *); /* call routine */
  vpiHandle    obj;                          /* trigger object */
  p_vpi_time   time;                         /* callback time */
  p_vpi_value  value;                        /* trigger object value */
  PLI_INT32    index;                        /* index of the memory word or
                                                var select that changed */
  PLI_BYTE8   *user_data;
} s_cb_data, *p_cb_data;

/*************************** CALLBACK REASONS ****************************/
/*************************** Simulation related **************************/
#define cbValueChange            1
#define cbStmt                   2
#define cbForce                  3
#define cbRelease                4

/***************************** Time related ******************************/
#define cbAtStartOfSimTime       5
#define cbReadWriteSynch         6
#define cbReadOnlySynch          7
#define cbNextSimTime            8
#define cbAfterDelay             9

/***************************** Action related ****************************/
#define cbEndOfCompile          10
#define cbStartOfSimulation     11
#define cbEndOfSimulation       12
#define cbError                 13
#define cbTchkViolation         14
#define cbStartOfSave           15
#define cbEndOfSave             16
#define cbStartOfRestart        17
#define cbEndOfRestart          18
#define cbStartOfReset          19
#define cbEndOfReset            20
#define cbEnterInteractive      21
#define cbExitInteractive       22
#define cbInteractiveScopeChange 23
#define cbUnresolvedSystf       24
```


Copyright 2000 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

```
/************************* Added with 1364-2000 *************************/
#define cbAssign                25
#define cbDeassign              26
#define cbDisable               27
#define cbPLIError              28
#define cbSignal                29

/************************ FUNCTION DECLARATIONS ************************/

/* callback related */
XXTERN vpiHandle  vpi_register_cb     PROTO_PARAMS((p_cb_data cb_data_p));
XXTERN PLI_INT32  vpi_remove_cb       PROTO_PARAMS((vpiHandle cb_obj));
XXTERN void       vpi_get_cb_info     PROTO_PARAMS((vpiHandle object,
                                                    p_cb_data cb_data_p));
XXTERN vpiHandle  vpi_register_systf  PROTO_PARAMS((p_vpi_systf_data
                                                      systf_data_p));
XXTERN void       vpi_get_systf_info  PROTO_PARAMS((vpiHandle object,
                                                    p_vpi_systf_data
                                                      systf_data_p));

/* for obtaining handles */
XXTERN vpiHandle  vpi_handle_by_name  PROTO_PARAMS((PLI_BYTE8 *name,
                                                    vpiHandle scope));
XXTERN vpiHandle  vpi_handle_by_index PROTO_PARAMS((vpiHandle object,
                                                    PLI_INT32 indx));

/* for traversing relationships */
XXTERN vpiHandle  vpi_handle          PROTO_PARAMS((PLI_INT32 type,
                                                    vpiHandle refHandle));
XXTERN vpiHandle  vpi_handle_multi    PROTO_PARAMS((PLI_INT32 type,
                                                    vpiHandle refHandle1,
                                                    vpiHandle refHandle2,
                                                    ... ));
XXTERN vpiHandle  vpi_iterate         PROTO_PARAMS((PLI_INT32 type,
                                                    vpiHandle refHandle));
XXTERN vpiHandle  vpi_scan            PROTO_PARAMS((vpiHandle iterator));

/* for processing properties */
XXTERN PLI_INT32  vpi_get             PROTO_PARAMS((PLI_INT32 property,
                                                    vpiHandle object));
XXTERN PLI_BYTE8 *vpi_get_str         PROTO_PARAMS((PLI_INT32 property,
                                                    vpiHandle object));

/* delay processing */
XXTERN void       vpi_get_delays      PROTO_PARAMS((vpiHandle object,
                                                    p_vpi_delay delay_p));
XXTERN void       vpi_put_delays      PROTO_PARAMS((vpiHandle object,
                                                    p_vpi_delay delay_p));

/* value processing */
XXTERN void       vpi_get_value       PROTO_PARAMS((vpiHandle expr,
                                                    p_vpi_value value_p));
XXTERN vpiHandle  vpi_put_value       PROTO_PARAMS((vpiHandle object,
                                                    p_vpi_value value_p,
                                                    p_vpi_time time_p,
                                                    PLI_INT32 flags));

/* time processing */
XXTERN void       vpi_get_time        PROTO_PARAMS((vpiHandle object,
                                                    p_vpi_time time_p));

/* I/O routines */
XXTERN PLI_UINT32 vpi_mcd_open        PROTO_PARAMS((PLI_BYTE8 *fileName));
```

 Copyright 2000 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

```
XXTERN PLI_UINT32 vpi_mcd_close       PROTO_PARAMS((PLI_UINT32 mcd));
XXTERN PLI_BYTE8 *vpi_mcd_name        PROTO_PARAMS((PLI_UINT32 cd));
XXTERN PLI_INT32  vpi_mcd_printf      PROTO_PARAMS((PLI_UINT32 mcd,
                                                    PLI_BYTE8 *format,
                                                    ...));
XXTERN PLI_INT32  vpi_printf          PROTO_PARAMS((PLI_BYTE8 *format,
                                                    ...));

/* utility routines */
XXTERN PLI_INT32  vpi_compare_objects PROTO_PARAMS((vpiHandle object1,
                                                    vpiHandle object2));
XXTERN PLI_INT32  vpi_chk_error       PROTO_PARAMS((p_vpi_error_info
                                                      error_info_p));
XXTERN PLI_INT32  vpi_free_object     PROTO_PARAMS((vpiHandle object));
XXTERN PLI_INT32  vpi_get_vlog_info   PROTO_PARAMS((p_vpi_vlog_info
                                                      vlog_info_p));

/* routines added with 1364-2000 */
XXTERN PLI_INT32  vpi_get_data        PROTO_PARAMS((PLI_INT32 id,
                                                    PLI_BYTE8 *dataLoc,
                                                    PLI_INT32 numOfBytes));
XXTERN PLI_INT32  vpi_put_data        PROTO_PARAMS((PLI_INT32 id,
                                                    PLI_BYTE8 *dataLoc,
                                                    PLI_INT32 numOfBytes));
XXTERN void      *vpi_get_userdata    PROTO_PARAMS((vpiHandle obj));
XXTERN PLI_INT32  vpi_put_userdata    PROTO_PARAMS((vpiHandle obj,
                                                    void *userdata));
XXTERN PLI_INT32  vpi_vprintf         PROTO_PARAMS((PLI_BYTE8 *format,
                                                    va_list ap));
XXTERN PLI_INT32  vpi_mcd_vprintf     PROTO_PARAMS((PLI_UINT32 mcd,
                                                    PLI_BYTE8 *format,
                                                    va_list ap));
XXTERN PLI_INT32  vpi_flush           PROTO_PARAMS((void));
XXTERN PLI_INT32  vpi_mcd_flush       PROTO_PARAMS((PLI_UINT32 mcd));
XXTERN PLI_INT32  vpi_control         PROTO_PARAMS((PLI_INT32 operation,...));
XXTERN vpiHandle  vpi_handle_by_multi_index PROTO_PARAMS((vpiHandle obj,
                                                    PLI_INT32 num_index,
                                                    PLI_INT32 *index_array));



/************************** GLOBAL VARIABLES **************************/
PLI_VEXTERN PLI_DLLESPEC void (*vlog_startup_routines[])();
  /* array of function pointers, last pointer should be null */


#undef PLI_EXTERN
#undef PLI_VEXTERN

#ifdef VPI_USER_DEFINED_DLLISPEC
#undef VPI_USER_DEFINED_DLLISPEC
#undef PLI_DLLISPEC
#endif
#ifdef VPI_USER_DEFINED_DLLESPEC
#undef VPI_USER_DEFINED_DLLESPEC
#undef PLI_DLLESPEC
#endif

#ifdef PLI_PROTOTYPES
#undef PLI_PROTOTYPES
#undef PROTO_PARAMS
#undef XXTERN
#undef EETERN
#endif
```


Copyright 2000 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

```
#endif /* VPI_USER_H */
```

 Copyright 2000 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

Annex H

Bibliography

(informative)

[B1] IEEE Std 754-1985 (Reaff 1990), IEEE Standard for Binary Floating-Point Arithmetic (ANSI).[1]
[B2] IEEE Std 1497-1999, SDF.

[1]IEEE publications are available from the Institute of Electrical and Electronics Engineers, 445 Hoes Lane, P.O. Box 1331, Piscataway, NJ 08855-1331, USA.

Copyright 2000 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

Copyright 2000 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.