Appendix B

This appendix describes the BNF for the assertion portion of the language.

BNF of Assertions

immediate_assertion is a statement_item. statement_item is defined in System Verilog.

concurrent_assertion_item can be a module_item or a statement_item. module_item is defined in System Verilog.

concurrent_assertion_item ::=
| property_declaration
| sequence_declaration
| concurrent_assert_statement
| concurrent_cover_statement

procedural_assertion_items ::=
concurrent_assert_statement
| concurrent_cover_statement
| immediate_assert_statement

immediate_assert_statement ::=
assert (expression **)** action_block

concurrent_assert_statement ::=
assert (** property_instance **) action_block
| **assert (** sequence_instance **)** action_block

concurrent_cover_statement ::=
cover (property_instance **)** statement_or_null
| **cover (** sequence_instance **)** statement_or_null

action_block ::=
statement_or_null [**else** statement_or_null]

statement_or_null ::=
statement
| **;**

property_declaration ::=
property property_identifier [property_formal_list] **;**
{ property_decl_item }
property_spec **;**
endproperty [**:** property_identifier]

property_formal_list ::=
(formal_list_item { , formal_list_item } **)**

property_declaration_item ::=
sequence_declaration
| variable_declaration
identifier [**(** identifier { **,** identifier } **)**] **=** prop_spec

property_spec ::=
[**disable iff (** expression **)**] [**not**] property_expression

property_expression ::=
sequence_spec
| property_implication
property_implication ::=
sequence_spec => [**not**] sequence_spec
| sequence_spec |=> [**not**] sequence_spec
property_instance::=
property_identifier [**(** actual_arg_list **)**]
sequence_declaration ::=
sequence sequence_identifier [sequence_formal_list] **;**
{ sequence_decl_item }
sequence_spec **;**
endsequence [**:** sequence_identifier]
sequence_formal_list ::=
(formal_list_item { , formal_list_item } **)**
sequence_decl_item ::=
variable_declaration
| sequence_declaration
sequence_spec ::=
sequence_phrase
sequence_phrase ::=
[cycle_delay_range] sequence_element { cycle_delay_range sequence_element }
cycle_delay_range ::=
constant_expression
| ## **[** const_range_expression **]**
sequence_element ::=
[event_control] sequence_element {**,** function_blocking_assign }
| expression [boolean_abbrev]
| sequence_expr
| sequence_instance [sequence_abbrev]
| **(** sequence_phrase **)** [sequence_abbrev]
sequence_expr ::=
sequence_element **and** sequence_element
| sequence_element **intersect** sequence_element
| sequence_element **or** sequence_element
| first_match (sequence_phrase **)**
| expression **throughout** sequence_element
| sequence_element **within** sequence_element
sequence_instance::=
sequence_identifier [**(** actual_arg_list **)**]
boolean_abbrev ::=
repeat_operator
| nth_event_operator

| counting_operator

sequence_abbrev ::=

repeat_operator

repeat_operator ::=

[* const_range_expression **]**

counting_operator ::=

[*= const_range_expression **]**

nth_event_operator ::=

[*> const_range_expression **]**

const_range_expression ::=

constant_expression

| constant_expression **:** constant_expression

| constant_expression **: $**

These operators are part of the built-in system functions and methods.

boolean_expr_op ::=

| seq_name.**ended**

| seq_name.**matched**

| value_change_functions

| **$past (** expression [**,** number_of_ticks] **)**

| **$countones (** expression **)**

value_change_functions::=

$rose (expression **)**

| **$fell (** expression **)**

| **$stable (**expression **)**

formal_list_item ::=

formal_identifier [**=** actual_arg_expr]

actual_arg_list ::=

(actual_arg_expr { **,** actual_arg_expr } **)**

| (.formal_identifier **(** actual_arg_expr **)** {**, .**formal_identifier **(** actual_arg_expr **)** } **)**

actual_arg_expr ::=

expression

| identifier

| event_control

| **[**constant_range_expression **: inf]**

| **$stable (**expression **)**

template_declaration ::=

template template_identifier [template_formal_list] **;**

template_body

endtemplate [**:** template_identifier]

template_body ::=

{ module_or_generate_item }

| { interface_or_generate_item }

| { statement }

template_formal_list ::=
    (formal_list_item { **,** formal_list_item } **)**
template_instance ::=
    *template*_identifier [actual_arg_list]**;**
bind_directive ::=
    bind module_instance_name program instantiation ;

