

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 ( property_spec ) action_block

concurrent_cover_statement ::=
    cover property ( property_instance ) statement_or_null

action_block ::=
    [statement] [ else statement ] ;

statement_or_null ::=
    statement
    | ;

property_declaration ::=
    property property_identifier [ property_formal_list ] ;
    { variable_declaration }
    { sequence_declaration }
    property_spec ;
    endproperty [ : property_identifier ]

property_formal_list ::=
    ( formal_list_item { , formal_list_item } )

property_modifier ::=
    disable iff ( expression ) [ not ]
    | [ disable iff ( expression ) ] not

property_spec ::=
    [ event_control ] property_modifier property_expr
    | property_expr
```

```

property_expression ::=
    sequence_spec
    | property_implication
property_implication ::=
    sequence_expr |-> [ not ] sequence_expr
    | clocked_sequence |-> [ not ] sequence_expr
    | sequence_expr |==> [ not ] sequence_expr
    | clocked_sequence |==> [ not ] sequence_expr
    | multi_clock_sequence |==> [ not ] multi_clock_sequence
property_instance ::=
    property_identifier [ ( actual_arg_list ) ]
sequence_declaration ::=
    sequence sequence_identifier [ sequence_formal_list ] ;
    { variable_declaration }
    { sequence_declaration }
    sequence_spec ;
    endsequence [ : sequence_identifier ]
sequence_formal_list ::=
    ( formal_list_item { , formal_list_item } )
sequence_spec ::=
    multi_clock_sequence
    | sequence_expr
multi_clock_sequence ::=
    clocked_sequence { ## clocked_sequence }
clocked_sequence ::=
    event_control sequence_expr
sequence_expr ::=
    [ cycle_delay_range ] sequence_expr { cycle_delay_range sequence_expr }
    | expression { , function_blocking_assign } [ boolean_abbrev ]
    | sequence_instance [ sequence_abbrev ]
    | ( sequence_expr ) [ sequence_abbrev ]
    | sequence_expr and sequence_expr
    | sequence_expr intersect sequence_expr
    | sequence_expr or sequence_expr
    | first_match ( sequence_expr )
    | expression throughout sequence_expr
    | sequence_expr within sequence_expr
sequence_instance ::=
    sequence_identifier [ ( actual_arg_list ) ]
cycle_delay_range ::=
    ## constant_expression
    | ## [ const_range_expression ]
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_or_range_expression ::=
        constant_expression
        | constant_range_expression
const_range_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 ::=
        event_expression
template_declaration ::=
        template template_identifier [( template_formal_list ) ] ;
        { concurrent_assertion_item }
        endtemplate [ : template_identifier ]
template_formal_list ::=
        formal_list_item { , formal_list_item }
formal_list_item ::=
        formal_identifier [ = actual_arg_expr ]

```

```
template_instance ::=  
    template_identifier [instance_name] [ ( actual_arg_list ) ]  
bind_directive ::=  
    bind module_instance_name program instantiation ;
```

Following precedence rules apply:

- ·boolean operators have higher precedence than sequence operators
- ·boolean operators follow system Verilog precedence
- ·comma for binding expression with assignment

·Precedence of operators in order (higher to lower) as below:

- 1) osystem verilog expression operators
- 2) comma for binding expression with assignment
- 3) [* , [*=, [*->
- 4) and , intersect
- 5) or
- 6) throughout
- 7) within
- 8) concatenation (## cycle_delay_range)

·Left to right association for operators