

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 ::=

```
    concurrent_assert_statement  
    | concurrent_cover_statement  
    | concurrent_assertion_item_declaration
```

concurrent_assertion_item_declaration ::=

```
    property_declaration  
    | sequence_declaration
```

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  
    | assert property ( property_instance ) action_block
```

concurrent_cover_statement ::=

```
    cover property ( property_spec ) statement_or_null  
    | cover property ( property_instance ) statement_or_null
```

action_block ::=

```
    statement_or_null  
    | [statement] else statement_or_null
```

statement_or_null ::=

```
    statement  
    | ;
```

property_declaration ::=

```
    property property_identifier [ property_formal_list ] ;  
        { assertion_variable_declaration }  
    property_spec ;  
    endproperty [ : property_identifier ]
```

property_formal_list ::=

```
    ( formal_list_item { , formal_list_item } )
```

property_spec ::=

```
    [ clocking_event ] [ disable iff ( expression ) ] [ not ] property_expr  
    | [ disable iff ( expression ) ] [ not ] multi_clock_property_expr
```

```

property_expr ::=
    sequence_expr
    | sequence_expr |-> [ not ] sequence_expr
    | sequence_expr |=> [ not ] sequence_expr
multi_clock_property_expr ::=
    multi_clock_sequence
    | multi_clock_sequence |=> [ not ] multi_clock_sequence
property_instance ::=
    property_identifier [ ( actual_arg_list ) ]
sequence_declaration ::=
    sequence sequence_identifier [ sequence_formal_list ] ;
    { assertion_variable_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 ::=
    clocking_event sequence_expr
sequence_expr ::=
    [ cycle_delay_range ] sequence_expr { cycle_delay_range sequence_expr }
    | ( expression { , function_blocking_assign } ) [ boolean_abbrev ]
    | 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 ::=
    consecutive_repetition
    | non_consecutive_repetition

```

```
    | goto_repetition
sequence_abbrev ::=
    consecutive_repetition
consecutive_repetition ::=
    [* const_range_expression ]
non_consecutive_repetition ::=
    [*= const_range_expression ]
goto_repetition ::=
    [*-> 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
assertion_variable_declaration ::=
    data_type list_of_variable_identifiers;
bind_directive ::=
    bind module_instance_name bind_instantiation ;
module_instance_name ::=
    module_identifier
    | name_of_instance
bind_instantiation ::=
    program_instantiation
    | module_instantiation
```

| interface_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) ·system 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