Insert as Section 8.10.1

8.10.1 Sequence events

A sequence instance can be used in event expressions to control the execution of procedural statements based on the successful completion of the sequence. This allows the endpoint of a named sequence to trigger multiple actions in other processes. Syntax 17-2 describes the syntax for declaring sequence instances and sequence expressions. A sequence instance can be used directly in an event expression, as shown below.

```
event_control ::=                                                          // from Annex A.6.5
                     @ event_identifier
                    | @ ( event_expression )
                    | @*
                    | @ (*)
                    | @ sequence_instance

 event_expression ::=
                       [ edge_identifier ] expression [ iff expression ]
                      | sequence_instance [ iff expression ]
                      | event_expression or event_expression
                      | event_expression , event_expression
```

Syntax 8-9—Event control and Event expression (excerpt from Annex A).

When a sequence instance is specified in an event expression, the process executing the event control shall block until the given sequence reaches its end-point, that is, the sequence succeeds non-vacuously. A sequence reaches its end point whenever there is a match for the entire sequence expression. When the process unblocks it shall continue to execute in the Reactive region following the end point.

An example of using a sequence as an event control is shown below.

```
sequence abc;
    @(posedge clk) a ##1 b ##1 c;
endsequence

program test;
    initial begin
        @ abc $display( "Saw a-b-c" );
        L1 : ...
    end
endprogram
```

In the example above, when the named sequence `abc` reaches its end point, the initial block in the program block test is unblocked, which then displays the string “Saw a-b-c” and continues execution with the statement labeled L1. In this case, the end of the sequence acts as the trigger to unblock the event.

A sequence used in an event control is instantiated (as if by an assert property statement); the event control is used to synchronize to the end of the sequence, regardless of its start-time. Sequence arguments shall be static; automatic variables used as sequence arguments shall result in an error.

Insert as new Section 8.11 (after Event control)

8.11 Level-sensitive sequence controls

The execution of procedural code can be delayed until a sequence termination status is true. This is accomplished using the level-sensitive **wait** statement in conjunction with the built-in method that returns the current end status of a named sequence: triggered.

The **triggered** sequence method evaluates to true if the given sequence has reached its end point at that particular point in time (in the current time-step), and false otherwise. The triggered status of a sequence is set during the Observe region and persists throughout the time-step (i.e., until simulation time advances).

For example:

```
sequence abc;
    @(posedge clk) a ##1 b ##1 c;
endsequence

sequence de;
    @(negedge clk) d ##[2:5] e;
endsequence

program check;
    initial begin
        wait( abc.triggered || de.triggered );
        if( abc.triggered )
            $display( "abc succeeded" );
        if( de.triggered )
            $display( "de succeeded" );
        L2 : ...
    end
endprogram
```

In the above example, the initial block in program check waits for the end point (success) of either sequence `abc` or sequence `de`. When either condition evaluates to true the **wait** statement unblocks the process, which displays the sequences that caused the process to unblock and then continues to execute the statement labeled L2.

Insert as Section 17.15

17.15 The expect statement

The expect statement is a procedural blocking statement that allows a property to be declared and also to wait for the first successful match of the property. The syntax of the expect statement is given below.

```
statement_item ::=                                                    // from Annex A.6.4
        …
      | {attribute_instance} expect ( property_spec ) [ action_block ]
```

Syntax 17-18: Expect statement (excerpt from Annex A)

The expect statement accepts the same syntax used to declare a property. An **expect** statement causes the executing process to block until the given property succeeds (i.e., the sequence reaches its end point) or fails. The **expect** statement unblocks at the earliest match of the property (i.e., first_match). The statement following the **expect** shall execute in the Reactive region following the success of the property, or the first failed attempt. In either case, the specified property terminates its evaluation when the process unblocks.

When executed, the **expect** statement automatically starts evaluating the given property on the subsequent clocking event, that is, the first attempt shall take place on the next clocking event. When the process unblocks (due to the property succeeding or failing), the property stops being evaluated. If the property fails at its clocking event, the optional **else** clause of the action block is executed. If the property succeeds the optional pass statement of the action block is executed.

The semantics of the **expect** statement are to block until first match (or failure) of the given property and whose starting time is greater than the time at which the expect statement executes.

```
program tst;
    initial begin
        # 200ms;
        expect( @(posedge clk) a ##1 b ##1 c ) else $error( "expect failed" );
        ABC: ...
    end
endprogram
```

In the above example, the expect statement blocks the first statement in the initial block of program tst until the sequence a -> b -> c is recognized starting with the following clocking event (posedge clk) after 200ms. If the sequence is matched at the corresponding time, the process is unblocked and continues to execute on the statement labeled ABC. If the sequence fails then the else clause is executed, which in this case generates a run-time error. For the expect above to succeed, the sequence a–>b–>c must occur on the clocking event (posedge clk) immediately after time 200ms. If a is false on the first clocking event after 200ms, the expect fails. If a is true on the first clocking event after 200ms and b is false one clocking event later, the expect will also fail.

The expect statement can be incorporated in any procedural code, including tasks or class methods. Because it is a blocking statement, the property expression may safely refer to automatic variables as well as static variables. For example, the task below waits between 1 and 10 cycles for the variable data to have a particular value, which is an automatic argument. The second argument is used to return the result of the expect, 1 for success and 0 for failure.

```
integer data;
...
task automatic wait_for( integer value, output bit success );
    expect( @(posedge clk) ##[1:10] data == value ) success = 1; else
        success = 0;
endtask

initial begin
    bit ok;
    wait_for( 23, ok );     // wait for the value 23
    ...
end
```