In Section 8.10.1 , page 94 - Changes in ~~red~~ and blue

A sequence instance can be used in event expressions to control the execution of procedural statements based on the successful ~~completion~~match of the sequence. This allows the endpoint of a named sequence to trigger multiple actions in other processes. Syntax 17-2 and 17-4 describe~~s~~ the syntax for declaring named sequences and sequence instances ~~and sequence expressions~~. A sequence instance can be used directly in an event expression, as shown in Syntax 8-9.

When a sequence instance is specified in an event expression, the process executing the event control shall block until the ~~given~~specified 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~~. A process resumes execution following the Observe region in which the end point is detected.

In Section 17.16 , page 267 - Changes in ~~red~~ and blue

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~~ waiting on a property evaluation. The syntax of the **expect** statement accepts a named property or a property declaration, and is given below.

```
expect_property_statement ::=                                   // from Annex A.2.10
    expect ( property_spec ) action_block
```

Syntax 17-18—expect statement syntax (excerpt from Annex A)

The **expect** statement accepts the same syntax used to assert a property. An **expect** statement causes the executing process to block until the given property ~~successor~~ succeeds or fails. ~~The **expect** statement unblocks at the earliest match of the property (i.e., first_match).~~ The statement following the **expect** is scheduled to execute after processing the Observe region ~~following the success of the property, or the first failed attempt~~in which the property completes its evaluation. When ~~In either case (i.e.,~~ the property succeeds or fails~~,), the specified property terminates its evaluation when~~ the process unblocks, and the property stops being evaluated (i.e., no property evaluation is started until that **expect** statement is executed again).

When executed, the **expect** statement ~~automatically~~ starts a single thread of evaluation for ~~evaluating~~ the given property on the subsequent clocking event, that is, the first ~~attempt~~evaluation 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 specifies a property that consists of the sequence `a ##1 b ##1 c`. The **expect** statement (second statement in the initial block of program `tst`) blocks until the sequence ~~a -> b -> c~~ `a ##1 b ##1 c` is ~~recognized~~matched, or determined not to match. The property evaluation starts on ~~starting wth~~ the ~~following~~ clocking event (posedge clk) ~~after~~following the 200ms delay. 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 to match then the else clause is executed, which in this case generates a run-time error. For the expect above to succeed, the sequence `a ##1 b ##1 c` must match starting 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 sequence will not match if `a`, `b`, or `c` are evaluated to be false at the first, second or third clocking event respectively.

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~~clock ticks for ~~the~~ variable `data` to ~~have~~ equal a particular value, which is specified by ~~an~~ automatic argument `value`. The second argument, `success`, is used to return the result of the **expect** statement:~~,~~ 1 for success and 0 for failure.