Modify Syntax of Section 8.10 (Event control) as shown

```
delay_or_event_control ::=
                        // from Annex A.6.5
         delay_control
        | event_control
        | repeat ( expression ) event_control

delay_control ::=
         # delay_value
        | # ( mintypmax_expression )

event_control ::=
         @ event_identifier
        | @ ( event_expression )
        | @*
        | @ (*)

event_expression ::=
         [ edge_identifier ] expression [ iff expression ]
        | event_expression or event_expression
        | event_expression , event_expression
        | begin hierarchical_btf_identifier
        | end hierarchical_btf_identifier

hierarchical_btf_identifier :: =
         hierarchical_task_identifier
        | hierarchical_function_identifier
        | hierarchical_block_identifier
        | hierarchical _identifier { class_identifier :: } method_identifier

edge_identifier ::= posedge | negedge
           // from Annex A.7.4
```

Syntax 8-8—Delay and event control syntax (excerpt from Annex A)

Add to the end of Section 8.10 (Event control)

SystemVerilog event expressions can be triggered by the start or the end of execution of a given named block, task, function, or class method. Event expressions that specify the **begin** keyword followed by a hierarchical identifier denoting a named block, task, function, or class method shall be triggered immediately before the corresponding block, task, function, or method begins executing its first statement. Event expressions that specify the **end** keyword followed by a hierarchical identifier denoting a named block, task, function, or class method shall be triggered immediately after the corresponding block, task, function, or method executes its last statement. Event expressions that specify the **end** of execution shall not be triggered if the block, task, function, or method is disabled.

For example:

```
task send_receive(inout byte b);
      bus <= b;
      # 5
      b = bus;
```

```
        endtask

    task check_sr();
        @( begin send_receive ) $display( "sent some data" );
        @( end send_receive ) $display( "received some data" );
    endtask
```

When task check_sr is called, it will block until task send_receive is called. The first line of task check_sr unblocks when a call to send_receive takes place. Likewise, the second line of task_sr will wait until the task send_receive terminates (i.e., the task returns).