```
Updated 01/16/2024
// PACKAGE UPDATE 12/07/2021
// https://verificationacademy.com/forums/systemverilog/sva-package-
dynamic-and-range-delays-and-repeats
/* 1) Attached are the .sv file, and simulation results with Questa.
2) // The package was verified through analysis of the sequences and
through random testing
// Sequences that use the first_match() function along with a range
repeat were reported
// not compatible with formal verification tools
 3) It would be great if 1800'2022 adds the dynamic possibilities for the
definition of the delays and repeats.    The value of those variables can
be captured and saved at the attempt cycle.    I didn't lock them in the
package.

BTW, these are complex sequences, and it took some time to get them to
that point. */

/* Ben Cohen Dec 7, 2021
// This file includes the delay and repeat package along with
// a testbench to demonstrate its application.
// The package was verified through analysis of the sequences and through
random testing
// Sequences that use the first_match() function along with a range
repeat were reported
// not compatible with formal verification tools */


package sva_delay_repeat_range_pkg;
    //-----------------------------------------------------------------
    // ******       DYNAMIC DELAY ##d1 **********
    // Implements     ##[d1]
    // Application: sq_1 ##0 dynamic_delay(d1) ##0 sq_2;
    sequence dynamic_delay(count);
        int v;
      (count<=0) or ((1, v=count) ##0 (v>0, v=v-1) [*0:$] ##1 v<=0);
    endsequence // dynamic_delay

    //-----------------------------------------------------------------
    // ******       DYNAMIC DELAY RANGE ##[d1:d2] **********
    // Implements     ##[d1:d2] ##0 a_sequence
    // for use in consequent
    // Application:  $rose(a)  |-> dynamic_delay_lohi_sq(d1, d2, q) ;
    /* sequence dynamic_delay_lohi_sq(d1, d2, sq);   // DO NOT USE
        int v1, vdiff;
          ( (1, v1=d1, vdiff=d2-d1) ##0 dynamic_delay(v1)   ##0
              (vdiff>=0, vdiff=vdiff - 1)[*1:$] ##0 sq);
    endsequence */

    sequence dynamic_delay_lohi_sq(d1, d2, sq);
      int v1, vdiff;
        dynamic_delay(d1)##0
        (sq or
        (1, vdiff=d2-d1) ##0 (vdiff>0, vdiff=vdiff - 1)[*1:$] ##1 sq);
```

```
    endsequence

    sequence dynamic_delay_fm_lohi_sq(d1, d2, sq);
            int v1, vdiff;
             first_match(dynamic_delay_lohi_sq(d1, d2, sq));
    endsequence // dynamic_delay_fm_lohi_sq

    //-----------------------------------------------------------------
    // ******       DYNAMIC REPEAT q_s[*d1] **********
    // Implements   a_sequence[*count]
    // Application:  $rose(a)  |-> sq_rpt_simple_count(sq_1, d1)
    sequence sq_rpt_simple_count(sq, count);
        int v=count;
        (1, v=count) ##0 ( v>0 ##0 sq, v=v-1) [*1:$] ##0 v<=0;
    endsequence // sq_rpt_simple_count
    //Note:  "The sequence_expr of a sequential property shall not admit
an empty   match (see 16.12.22)."

    //-----------------------------------------------------------------
    // ******       DYNAMIC REPEAT RANGE sq_1[*range] ##1 b] **********
    // Implements   a_sequence[*1:range] b_sequence
    // Application:  $rose(a) |-> sq1_rpt_simple_range_sq2(sq_1, d1,
sq2_2);
    //NOTE: RANGE MUST BE GREATER THAN ZERO !!!!!!
    sequence sq1_rpt_simple_range_sq2(sq1, range,  sq2);
      int v, diff;
      (range>0,  diff=range)  ##0
      ((diff>0  ##0 sq1, diff=diff-1'b1) [*1:$] ##1 sq2);
    endsequence


    sequence sq1_rpt_simple_range_fm_sq2(sq1, range,  sq2);
      int v, diff;
      (range>0,  diff=range)  ##0
      first_match((diff>0  ##0 sq1, diff=diff-1'b1) [*1:$] ##1 sq2);
    endsequence


    //-----------------------------------------------------------------
    // ******       DYNAMIC REPEAT LO-HI RANGE  **********
    // Implements  a_sequence[*lo:hi] ##1 b_sequence
    // Application:  $rose(a)|-> sq1_rpt_lohi_sq2(sq_1, d1, d2, sq_2);
    sequence  sq1_rpt_lohi_sq2(sq1, d1, d2, sq2);
        int v, diff;
        (1, v=d1-1, diff=d2-d1+1) ##0
        sq_rpt_simple_count(sq1, v) ##1
        // TBD Ben Srini q1_rpt_simple_range_q2(sq1, diff, sq2);
        sq1_rpt_simple_range_sq2(sq1, diff, sq2);
    endsequence // sq1_rpt_lohi_sq2

endpackage // sva_delay_repeat_range_pkg


// ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
```

```
// ++++++++++++++++++ testbench +++++++++++++++++++++
import uvm_pkg::*; `include "uvm_macros.svh"
import sva_delay_repeat_range_pkg::*;
module top;
    timeunit 1ns;  timeprecision 100ps;
    bit clk, a, b, c=1, w;
    int d1=2, d2=5, range=7;
    sequence sq_a1_2_b; a ##[1:2] b; endsequence
    sequence sq_c1_4_b; c ##[1:4] b;        endsequence
    sequence sq_2w;  ##2 w; endsequence
    // default clocking @(posedge clk); endclocking
    initial forever #5 clk=!clk;

    // ******       DYNAMIC DELAY ##d1 **********
    ap_dyn_delay2: assert property(@ (posedge clk)
       $rose(a) |-> dynamic_delay(d1) ##0 sq_2w);

    ap_fix2_delay2: assert property(@ (posedge clk)
       $rose(a) |-> ##2 sq_2w);

    ap_dyn_delay0: assert property(@ (posedge clk)
       $rose(a) |-> dynamic_delay(1'b0) ##0 sq_2w);

    ap_fix_delay0: assert property(@ (posedge clk)
       $rose(a) |-> ##0 sq_2w);
//-----------------------------------------------------------------
    // ******       DYNAMIC DELAY RANGE ##[d1:d2] **********
    ap_dly_2_5_consequent_rng: assert property(@ (posedge clk)
      $rose(a) |-> dynamic_delay_lohi_sq(d1, d2, sq_c1_4_b));

    ap_fix_2to5_consequent: assert property(@ (posedge clk)
      $rose(a) |-> ##[2:5] sq_c1_4_b );

    ap_dly_rng_2_5_antc: assert property(@ (posedge clk)
     $rose(a) ##0 dynamic_delay_fm_lohi_sq(d1, d2, sq_c1_4_b) |-> sq_2w);

    ap_fix_2to5_antc: assert property(@ (posedge clk)
      first_match($rose(a) ##0 (##[2:5] sq_c1_4_b)) |-> sq_2w);

    //===
    // The sub-sequence (vdiff>=0, vdiff=vdiff - 1)[*1:$] ##0 sq)
    // causes the display of the failure to occur 1 cycle later
    // when vdiff==0 and sq is a nomatch.
    // At next cycle, vdiff==-1 and the term vdiff>0 is false.
    // That's when the error is display (1 cycle laters)
    // The passes are displayed at the correct cycle.
    ap_dly_2to2_consequent_rng: assert property(@ (posedge clk) // **
FLAG, but OK
      $rose(a) |-> dynamic_delay_lohi_sq(d1, 2, sq_c1_4_b));

    ap_fix_2to2_consequent: assert property(@ (posedge clk)
      $rose(a) |-> ##[2:2] sq_c1_4_b );
```

```
    ap_dly_0to0_consequent_rng: assert property(@ (posedge clk)  // **
FLAG, but OK
      $rose(a) |-> dynamic_delay_lohi_sq(0, 0, sq_c1_4_b));

    ap_fix_0to0_consequent: assert property(@ (posedge clk)
      $rose(a) |-> ##[0:0] sq_c1_4_b );

    //-----------------------------------------------------------------
    // ******       DYNAMIC REPEAT sq_a1_2_b[*d1] **********
    ap_rpt_count_2_cons: assert property(@ (posedge clk)
      $rose(a)|-> sq_rpt_simple_count(sq_a1_2_b, d1)  ##1 sq_2w);

    ap_rpt_2_fix_cons: assert property(@ (posedge clk)
       $rose(a)|-> sq_a1_2_b[*2] ##1 sq_2w);

    ap_rpt_count_2_antc: assert property(@ (posedge clk)
      $rose(a) ##0 sq_rpt_simple_count(sq_a1_2_b, d1) |->  ##1 sq_2w);

    ap_rpt_2_fix_antc: assert property(@ (posedge clk)
        $rose(a) ##0 sq_a1_2_b[*2] |-> ##1 sq_2w);

// ******       DYNAMIC REPEAT RANGE sq_1[*rang] ##1 b] **********
    ap_rpt_rng2_5_antc_all: assert property(@ (posedge clk)
      $rose(a)  ##1 sq1_rpt_simple_range_sq2(sq_a1_2_b, range, sq_c1_4_b)
                          |->   sq_2w);
    ap_rpt_fix2_5_antc_all: assert property(@ (posedge clk)
      $rose(a)  ##1  (sq_a1_2_b[*1:7] ##1 sq_c1_4_b) |->   sq_2w);

    ap_rpt_rng2_5_cons: assert property(@ (posedge clk)
      $rose(a) |-> ##1 sq1_rpt_simple_range_sq2(sq_a1_2_b, range,
sq_c1_4_b));

    ap_rpt_fix2_5_cons: assert property(@ (posedge clk)
      $rose(a) |-> ##1  (sq_a1_2_b[*1:7] ##1 sq_c1_4_b));

// first_match()
    ap_rpt_rng2_5_cons_all_fm: assert property(@ (posedge clk)
      $rose(a)  |-> sq1_rpt_simple_range_fm_sq2(sq_a1_2_b, range,
sq_c1_4_b) ##0   sq_2w);

    ap_rpt_range_fix2_5_cons_fm: assert property(@ (posedge clk)
      $rose(a)  |-> first_match(sq_a1_2_b[*1:7] ##1 sq_c1_4_b)   ##0
sq_2w);

    ap_rpt_rng2_5_antc_fm: assert property(@ (posedge clk)
      $rose(a)  ##1 sq1_rpt_simple_range_fm_sq2(sq_a1_2_b, range,
sq_c1_4_b));

    ap_rpt_fix2_5_antc_fm: assert property(@ (posedge clk)
      $rose(a)  ##1 first_match( (sq_a1_2_b[*1:7] ##1 sq_c1_4_b)));

    initial begin

      repeat(2) @(posedge clk);
```

```
      a <= 1'b0; b<=1'b0; c<=1'b0; w<=1'b0;
      @(posedge clk);
      a <= 1'b1; b<=1'b1; c<=1'b1; w<=1'b1;
      @(posedge clk) d1<=0; d2<=0;  // <------------------ ZERO test
      repeat(2) @(posedge clk);
      a <= 1'b0; b<=1'b0; c<=1'b0; w<=1'b0;
      @(posedge clk);
      a <= 1'b1; b<=1'b1; c<=1'b1; w<=1'b1;
      // Regular test
      @(posedge clk) d1<=2; d2<=5;  // <--------------------- Range test
      repeat(20) @(posedge clk);
        repeat(1000) begin
            @(posedge clk); #2;
            if (!randomize(a, b, c, w)  with
            { a dist {1'b1:=1, 1'b0:=1};
            b dist {1'b1:=1, 1'b0:=1};
            c dist {1'b1:=1, 1'b0:=1};
            w dist {1'b1:=1, 1'b0:=1}; }) `uvm_error("MYERR", "randomize
error");
        end
        #1;
        repeat(1500) begin
            @(posedge clk); #2;
            if (!randomize(a, b, c, w)  with
            { a dist {1'b1:=1, 1'b0:=2};
            b dist {1'b1:=3, 1'b0:=2};
            c dist {1'b1:=1, 1'b0:=1};
            w dist {1'b1:=3, 1'b0:=1}; }) `uvm_error("MYERR", "randomize
error");
        end
        repeat(1500) begin
            @(posedge clk); #2;
            if (!randomize(a, b, c, w)  with
            { a dist {1'b1:=3, 1'b0:=2};
            b dist {1'b1:=1, 1'b0:=2};
            c dist {1'b1:=4, 1'b0:=1};
            w dist {1'b1:=2, 1'b0:=1}; }) `uvm_error("MYERR", "randomize
error");
        end
        $stop;
    end
endmodule
```