1.1 Passing unbounded range as an argument

This proposal enhances the capability of specifying range that can be fully parameterized,i.e is passed as an argument to properties and sequences. Currently, the range is specified as

cycle_delay_const_range_expression ::=

constant_expression **:** constant_expression

| constant_expression **: $**

The symbol used to specify an unbounded range is $. The first constant_expression specifies the low range limit, while the second constant_expression specifies the high range limit. Since the high limit can be a $, it cannot be passed as an argument to a property. For example,

```
property inq(r1,r2);
   @(posedge clk) a ##[r1:r2] b ##1 c |=> d;
endproperty
assert inq(3, 10);
```

In the example above, r1 and r2 replace the range for the delay. However, in order to make the high range unbounded, the property must be rewritten as,

```
property inq1(r1);
   @(posedge clk) a ##[r1:$] b ##1 c |=> d;
endproperty
assert inq1(3);
```

This inability to make unbounded range as an argument causes difficulty in writing re-usable assertions.

This proposal allows $ as a symbolic constant representing unbounded integer value. It can be passed as a parameter using the parameter assignment mechanism, or as an argument to a property or a sequence. In addition, a compile time system function is provided to test whether a constant is a $. The syntax of the system function is

$isunbounded(const_expression);

This function returns true if the constant_expression is unbounded. $isunbounded can be used to conditionally generate properties using the generate statement.

It is also recommended to define basic arithmetic operations on $ to evaluate expression when $ is used in a constant expression.

In order minimize the impact on the evaluation of general expressions, restrictions must be placed on the use of $. $ may not be assigned to a variable, or passed as an argument to a function or a task.

The example below illustrates the benefit of using $ in writing properties concisely where the range is parameterized. The checker in the example ensures that a bus driven by signal `en` remains 0, i.e, quiet for the specified minimum (`min_quiet`) and maximum (`max_quiet`) quiet time.

Also note that function $isunbounded is used for checking the validity actual arguments.

```
interface quiet_time_checker( clk, reset_n, en);
   input reset_n;
   input clk;
   input [1:0] en;
   parameter min_quiet = 0;
   parameter max_quiet = 0;

   generate
      if ( max_quiet == 0) begin
```



```
            property quiet_time;
               @(posedge clk) reset_n |-> ($countones(en) == 1);
            endproperty
            a1: assert property (quiet_time);
         end
         else begin
            property quiet_time;
               @(posedge clk)
                  (reset_n && ($past(en) != 0) && en == 0)
                  |->(en == 0)[*min_quiet:max_quiet]
                     ##1 ($countones(en) == 1);
            endproperty
            a1: assert property (quiet_time);
         end
         if ((min_quiet == 0) && ($isunbounded(max_quiet))
            $display(warning_msg);
      endgenerate
   endinterface

   quiet_time_checker #(0, 0) quiet_never (clk,1,enables);
   quiet_time_checker #(2, 4) quiet_in_window (clk,1,enables);
   quiet_time_checker #(0, $) quiet_any (clk,1,enables);
```

Another example below illustrates that by testing for $, a property can be configued according to the requirements. When parameter `max_cks` is unbounded, it is not required to test for `expr` to become false.

```
   interface width_checker(clk, reset_n, expr);
      input clk;
      input reset_n;
      input expr;

      parameter min_cks = 1;
      parameter max_cks = 1;

      generate  begin
         if ($isunbounded(max_cks)) begin
            property width;
               @(posedge clk)
                  (reset_n && $rose(expr)) |-> (expr [* min_cks]);
            endproperty
            a2: assert property (width);
         end
         else begin
            property assert_width_p;
               @(posedge clk)
                  (reset_n && $rose(expr)) |-> (expr[* min_cks:max_cks])
                                               ##1 (!expr);
            endproperty
            a2: assert property (width);
         end
      endgenerate
   endinterface

   width_checker #(3, $) max_width_unspecified (clk,1,enables);
   width_checker #(2, 4) width_specified (clk,1,enables);
```


