For compatibility to 1800-2009, the following related functions are also provided and have equivalence to $countbits. Specifically,

- $countones(expression) is equivalent to $countbits(expression,'1).
- $onehot(expression) returns true if $countbits(expression,'1)==1, otherwise it returns false.
- $onehot0(expression) returns true if $countbits(expression,'1)<=1, otherwise it returns false.
- $isunknown(expression) returns true if $countbits(expression,'x','z')!=0, otherwise it returns false.

**Application example**: Write an assertion for a 16 bit input where there could be multiple requests, but there is a single grant with bits of higher order having higher priority.

The key to this assertion is to use a generate statement and assert that if a bit of the request vector (i.e., req[i]) is set, and if the value of the request vector (req) is less the index +1 (i +1'1), then the grant for that index (grnt[i]) should be set, and then a single bit of grant is asserted (i.e., $onehot(grnt)). Below is the model and a simple testbench for the assertion (see Chapter 9 for more information on writing assertions for testbenches).

```verilog
import uvm_pkg::*; `include "uvm_macros.svh"

module arbiter;                 // /Ch4/4.2/arbiter.sv
    bit[15:0] req, grnt;
    bit clk;
    initial forever #5 clk=!clk;
    generate for (genvar i=0; i<=15; i++)
        begin
            property p_arbiter;
                bit[16:0] v;
                (req[i]==1'b1, v=0, v[i+1]=1'b1) ##0 req < v |->
                    grnt[i]==1'b1 ##0 $onehot(grnt);
            endproperty : p_arbiter
            ap_arbiter: assert property(@(posedge clk) p_arbiter);
        endgenerate
    ap_zero_req0: assert property(@(posedge clk) req==0 |-> grnt==0);
    ap_zero_req1: assert property(@(posedge clk) req==1 |-> grnt==1);

    always @(posedge clk)
        if (!randomize(req, grnt) with {req <15; $onehot(grnt)==1'b1;})
            `uvm_error("MYERR", "This is a randomize error");
endmodule
```

### 4.2.3 Severity-level system functions

#### 4.2.3.1 SystemVerilog severity levels

A good verification environment provides messages of varied nature such as:

- Informative messages (e.g., debug messages, notes, ..)
- Warnings
- Errors
- Fatal Errors

SystemVerilog provides some system functions to classify the messages generated from assertions; these can also be used in general SystemVerilog code. They are classified according to their severity levels. By default, the severity of an assertion failure is "error". The standard requires that minimally the following information to be provided: