Verilog - Representation of Number Literals

”....... And here there be monsters!” (Capt. Barbossa)

Numbers are represented as:
<size>’<signed><radix>value (”<>” indicates optional part)

size  The number of binary bits the number is comprised of. Not the number of hex or decimal digits. Default is 32 bits.

’  A separator, single quote, not a backtick

signed  Indicates if the value is signed. Either s or S can be used. Not case dependent
        Default is unsigned.

radix  Radix of the number
        ’b or ’B : binary
        ’o or ’O : octal
        ’h or ’H : hex
        ’d or ’D : decimal
        default is decimal
Possible values for "value" are dependent on the radix

<table>
<thead>
<tr>
<th>Format</th>
<th>Prefix</th>
<th>Legal characters</th>
</tr>
</thead>
<tbody>
<tr>
<td>binary</td>
<td>'b</td>
<td>01xXzZ_?</td>
</tr>
<tr>
<td>octal</td>
<td>'o</td>
<td>0-7xXzZ_?</td>
</tr>
<tr>
<td>decimal</td>
<td>'d</td>
<td>0-9_</td>
</tr>
<tr>
<td>hexadecimal</td>
<td>'h</td>
<td>0-9a-fA-FxXzZ_?</td>
</tr>
</tbody>
</table>

The underscore "_" is a separator used to improve readability e.g.: 0010_1010_1110_0101 is easily read as 0x2AE5

- The character "x" or "X" represents unknown
- The character "z" or "Z" represents high impedance
- The character "?" or "?" same as Z (high impedance)
- The character "?" is also "don’t care" to synthesis
Verilog - Representation of Number Literals (cont.)

- If prefix is preceded by a number, number defines the bit width
- If no prefix given, number is assumed to be 32 bits
- Verilog expands `<value>` to fill given `<size>` working from LSB to MSB.
- If `<size>` is smaller than ”value”
  - MSB’s of ”value” are truncated with warning (tool dependent)
- If `<size>` is larger than ”value”
  - MSB’s of ”value” are filled
- Regardless of MSB being 0 or 1, 0 filling is done

<table>
<thead>
<tr>
<th>Left-most Bit</th>
<th>Expansion</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0 extend</td>
</tr>
<tr>
<td>1</td>
<td>0 extend</td>
</tr>
<tr>
<td>x X</td>
<td>x or X extend</td>
</tr>
<tr>
<td>z Z</td>
<td>z or Z extend</td>
</tr>
</tbody>
</table>
Some Examples:

```verilog
reg [7:0] v = 8'b1011; initial $displayb("v signed =\t", v);
  //v = 00001011, MSBs filled with zeros

reg [7:0] w = 3'b1011; initial $displayb("w signed =\t", w);
  //w = 00000011, bit 3 truncated then 0 filled
  //generates Modelsim compile warning (Redundant digits in numeric literal)
  //Runs without warning or error
```
Literal numbers may be declared as signed: `4’shF`

- 4 bit number (1111) interpreted as a signed 2s complement value
- Decimal value is -1.

Signed values are not necessarily sign extended because the sign bit is the MSB of the size, not the MSB of the value.

```
8’hA //unsigned value extends to: 00001010
8’shA //signed value extends to: 00001010
```

If the MSB of the size is one and is signed, sign extension will occur.

```
reg [11:0] p1 = 4’shA; initial $displayb ("p1 signed =\t", p1);
    //p1 = 1111_1111_1010, bit 3 is the sign bit

reg [11:0] p2 = 5’shA; initial $displayb ("p2 signed =\t", p2);
    //p2 = 0000_0000_1010, bit 3 was the sign bit, but was lost in extension
```

When the value is assigned to a bigger vector, the sign indication `<s>`, will force sign extension when the MSB of value is one. If a signed number such as 9’shA6, (8 bits in 9 bit vector) is assigned to a bigger vector the sign bit is lost and is not sign extended. Beware!
Literal numbers can also carry a sign: -4'sd15
This is equivalent to -(4'sd15) or -(1) or 1.

We need to be careful with how an explicit sign is interpreted. See the examples to follow.
Verilog - Representation of Number Literals (cont.)

Some examples:

module number_test;
  reg [11:0] a = 8’shA6; initial $displayb ("a=", a);
  // number 0xA6 is signed, MSB of size (7) is one, so its negative
  // so to produce the 12 bit result, its sign extended with 1’s, thus
  // a=1111_1010_0110

  reg [11:0] b = 8’sh6A; initial $displayb ("b=", b);
  // signed number 0x6A has MSB (7) is zero, its positive
  // so to produce the 12 bit result, its sign extended with 0’s, thus
  // b=0000_0110_1010

  reg [11:0] c = ’shA6; initial $displayb ("c=", c);
  //c is the signed number A6, but its MSB is zero as its 32 bits long
  // c=0000_1010_0110, not sign extended

  reg [11:0] d = ’sh6A; initial $displayb ("d=", d);
  //signed, unsized number 6A has MSB (31) zero so its positive:
  // 0000_0000_0000_0000_0000_0000_0110_1010
  //assign the 32 bit value to 12 bits:
  // d=0000_0110_1010
Some more examples:

```verilog
testbench #1: 8-bit signed
reg [11:0] e = -8'hA6; initial $displayb("e=", e);
// 0xA6 is signed, expanded to 8 bits with MSB (7) one: 1010_0110
// negating this with a minus sign: (2’s compliment) : 0101_1010
// now assign with sign extension:
// i.e.; -(8’hA6)
// e=0000_0101_1010

reg [11:0] f = -’hA6; initial $displayb("f=", f);
// 0xA6 is signed, unsized, with MSB (31) zero, so its positive:
// 0000_0000_0000_0000_0000_0000_1010_0110
// taking twos complement we get:
// 1111_1111_1111_1111_1111_1111_0101_1010
// assigning to 12 bits (by truncation) we get
// f=1111_0101_1010

reg [11:0] g = 9’hA6; initial $displayb("g=", g);
// 0xA6 is signed with MSB (8) zero, so its positive: 0_1010_0110
// assign it to a 12-bit reg by sign extending:
// g=0000_1010_0110

reg [11:0] h = 9’h6A; initial $displayb("h=", h);
// 0x6A is signed with MSB (8) zero, so its positive: 0_0110_1010
// assign it to a 12-bit reg by sign extending:
// h=0000_0110_1010
```