Motivation

Bit vector functions do not handle 4-valued type data adequately. Consider the following example: no more than one bit of the vector `v` may be simultaneously active (= high). This statement may be interpreted in several different ways when `v` is of type **`logic`**:

1) All X and Z bits are treated as zeros. `$onehot0(v)` will do the job. However, this is usually not a desired behavior since one cannot claim that the bits of `v` are mutually exclusive if, for example, there is one bit set to 1 and one or more bits having value X.

2) If at least one bit of v has value X or Z then decide that the bits of v cannot be mutually exclusive. This may be expressed as `$onehot0(v) && !$isunknown(v)`. This definition is usually overkill since if at most one bit has an unknown value and all other bits are zero then two bits cannot be simultaneously active.

3) At most one bit of `v` has a non-zero value (i.e., 1, X, or Z). This interpretation is one of the most useful. This condition may be expressed as `$countones(~v) >= $bits(v) - 1`. This solution is clumsy.

4) At most one bit of v has value 1 or X, other bits should have values Z. This definition may be useful for buses. It is difficult to express this condition in a satisfactory way.

To overcome existing limitations we propose to introduce finer grained bit vector functions useful in most common cases.

Suggested Solution

16.12 System functions

REPLACE

Assertions are commonly used to evaluate certain specific characteristics of a design implementation, such as whether a particular signal is “one-hot”. The following system functions are included to facilitate such common assertion functionality:

— `$onehot (<expression>)` returns true if only 1 bit of the expression is high.

— `$onehot0 (<expression>)` returns true if at most 1 bit of the expression is high.

— `$isunknown (<expression>)` returns true if any bit of the expression is `X` or `Z`. This is equivalent to `^(<expression>) === 1'bx`.

All of the above system functions have a return type of **`bit`**. A return value of `1’b1` indicates true, and a return value of `1’b0` indicates false.

Another useful function provided for the Boolean expression is `$countones`, to count the number of ones in a bit vector expression.

```
$countones ( expression)
```

A bit with value X or Z is not counted towards the number of ones.

WITH

Assertions are commonly used to evaluate certain specific characteristics of a design implementation, such as whether a particular signal is “one-hot”. The following system functions are included to facilitate such common assertion functionality:

- `$onehot (<expression>)` returns true if only 1 bit of the expression ~~is high~~ has value 1.
- `$onehot_4 (<expression>)` returns true if only 1 bit of the expression has value 1, and all other bits of the expression are 0.
- `$onecold (<expression>)` returns true if only 1 bit of the expression has value 0.
- `$onecold_4 (<expression>)` returns true if only 1 bit of the expression has value 0, and all other bits of the expression are 1.
- `$onehot0 (<expression>)` returns true if at most 1 bit of the expression ~~is high~~ has value 1.
- `$onehot0_4 (<expression>)` returns true if at most 1 bit of the expression has value 1, X or Z.
- `$onehot0_4Z (<expression>)` returns true if at most 1 bit of the expression has value 1, and all other bits have value Z.
- `$isunknown (<expression>)` returns true if any bit of the expression is `X` or `Z`. This is equivalent to `^(<expression>) === 1'bx`.

All of the above system functions have a return type of **`bit`**. A return value of `1’b1` indicates true, and a return value of `1’b0` indicates false.

Another useful group of functions ~~function~~ provided for the Boolean expression is `$countones`, to count the number of ~~ones~~ bits having specific values in a bit vector expression.

~~`$countones ( expression)`~~

~~A bit with value X or Z is not counted towards the number of ones.~~

- `$countones (<expression>)` returns the number of bits in expression having values 1.
- `$count1XZ (<expression>)` returns the number of bits in expression having values 1, X, or Z.
- `$countzeros (<expression>)` returns the number of bits in expression having values 0.
- `$count0XZ (<expression>)` returns the number of bits in expression having values 0, X, or Z.
- `$countunknowns (<expression>)` returns the number of bits in expression having values X or Z.
- `$countX (<expression>)` returns the number of bits in expression having values X.
- `$countZ (<expression>)` returns the number of bits in expression having values Z.