INSERT after 7.15

7.16 Operator Overloading

There are various kinds of arithmetic that can be useful: saturating, arbitrary size floating point, carry save etc. It is convenient to use the normal arithmetic operators for readability, rather than relying on function calls.

The overload declaration allows the arithmetic operators to be applied to data types that are normally illegal for them, such as unpacked structures. It does not change the meaning of the operators for those types where it is legal to apply them. This means that such code does not change behavior when operator overloading is used.

overload_declaration ::=
function overload_operator data_type function_identifier **(**list_of_dpi_proto_formals**)**

overload_operator ::= **+** | **++** | **-** | **--** | ***** | ****** | **/** | **%** | **==** | **!=** | **<** | **<=** | **>** | **>=** | **=**

The overload declaration links an operator to a function prototype. The arguments are matched, and the type of the result is then checked. Multiple functions may have the same arguments and different return types; however, a cast must be used to select the correct function when used in a self-determined context. For example, suppose there is a structure type float:

```
typedef struct {
      bit sign;
      bit [10:0] mantissa;
      bit [3:0] exponent'
} float;
```

The + operator can be applied to this structure by invoking a function as indicated in the overloading declarations below:

```
function + float faddif(int, float);
function + float faddif(shortint, float);
function + float faddif(byte, float);
function + float faddfi(float, int);
function + float faddfi(float, shortint);
function + float faddfi(float, byte);
function + float faddrf(real, float);
function + float faddrf(shortreal, float);
function + float faddfr(float, real);
function + float faddfr(float, shortreal);
function + float faddff(float, float);
function + float fcopyf(float); // unary +
function + float fcopyi(int); // unary +
function + float fcopyi(shortint); // unary +
function + float fcopyi(byte); // unary +
function + float fcopyr(real); // unary +
function + float fcopyr(shorteal); // unary +

float A, B, C, D;
assign A = B + C;   // equivalent to A = faddff(B, C);
assign D = A + 1.0; // equivalent to A = faddfr(A, 1.0);
```

The overloading declaration links the operator to each function prototype according to the equivalent argument types in the overloaded expression, which must match exactly. Note that the function prototype does not need to match the actual function declaration exactly. If it does not, then the normal implicit casting rules apply when calling the function.

Overloading the assignment operator also serves to overload implicit assignments or casting. Similarly, the assignment operator can be overloaded using the same functions as the unary +, as shown below:

```
function = float fcopyf(float);
function = float fcopyi(int); // cast int to float
function = int   ficopy(float); // cast float to int
function = real  frcopy(float); // cast float to real
function = float fcopyi(shortint); // unary
function = float fcopyi(byte); // unary
function = float fcopyr(real); // unary
function = float fcopyr(shorteal); // unary

float A;
final $display("The final value of A is %g", real'(A)); // must cast
```

The operators that can be overloaded are the arithmetic operators, the relational operators and assignment. No format can be assumed for 0 or 1, so the user cannot rely on subtraction to give equality, or on addition to give increment. Similarly no format can be assumed for positive or negative, so comparison must be explicitly coded.

An assignment operator such as += is automatically built from both the + and = operators successively. For example

```
float A, B;
function = float fcopyf(float);
function + float faddff(float,float);
always @(posedge clk)
      A += B; // equivalent to A = fcopyf(faddff(A,B));
```

The scope and visibility of the overload declaration follows the same search rules as a data declaration. The overload declaration must be defined before use in a scope which is visible. The function bound by the overload declaration uses the same scope search rules as a function enable from the scope where the operator is invoked.

BNF changes
Add to 1.5

module_or_generate_item_declaration ::=
	…
	| overload_declaration

Add to 2.8

block_item_declaration ::=
  …
  | overload_declaration

overload_declaration ::=
  function overload_operator data_type function_identifier **(**list_of_dpi_proto_formals**)**

overload_operator ::= + | ++ | **-** | **--** | * | ** | / | **%** | == | **!**= | < | <= | > | >= | =