Mantis 3398

SV-DC User defined nets and resolution functions

Version 3

Change notes for the reader:

This version incorporates consensus agreements from the 4/06/2011 meeting related to initialization and tightened requirements for pure resolution functions.

At the end of 6.6 Net types, ADD,

6.6.7 User-defined net types

A user-defined nettype allows users to describe more general abstract values for a wire, including its resolution function. This nettype is similar to a typedef in some ways, but shall only be used in declaring a net. It provides a name for a particular datatype and optionally an associated resolution function.

net_type_declaration ::= nettype data_type net_type_identifier
(with [package_scope|class_scope] tf_identifier);

A net declared with that nettype therefore uses that datatype and when specified, the associated resolution functions. A user-defined net type has no default type; a specified data type is required. The data type shall include only integral types and real types as well as unpacked structs and fixed size arrays of a legal datatype.

A net declared with a user-defined nettype shall be called an atomic net. While an atomic net may have a composite value, each atomic net is intended to describe a single connection point in the design.

The resolution for a user-defined nettype is specified using a SystemVerilog function declaration. If a resolution function is specified, then when a driver of the net changes value, an update event is scheduled on the net in the active (or reactive) region. When the update event matures, the simulator calls the resolution function to compute the value of the net from the values of the drivers. The return type of the function shall match the data type of the nettype. The function shall accept an arbitrary number of drivers, since different instances of the net could be connected to different numbers of drivers. Any change in the value to one or more of the the drivers shall trigger the evaluation of the resolution function associated with that nettype.
A user defined resolution function for an atomic net with a data type T shall be an automatic function with a return type of T and a single input argument whose type is a dynamic array of type T. While a class function method may be used for a resolution function, such functions shall be class static methods as the method call occurs in a context where no class object is involved in the call. Parameterized variants of such methods can be created through the use of parameterized class methods as described in 13.8. [Note to editor: 13.8 is a new section in Mantis 696 and should be integrated first; any change to that section number should be reflected here.]

Two different nettypes can use the same datatype, but have different resolution functions. A nettype may be declared without a resolution function, in which case it shall be an error for a net of that nettype to have multiple drivers.

Due to non-determinism within scheduling regions, if there are multiple driver updates with a scheduling region, there may be multiple evaluations of the resolution function. A resolution function shall be automatic (or preserve no state information) and have no side effects. A resolution function shall not resize the dynamic array input argument nor shall it write to any part of the dynamic array input argument.

A force statement can override the value of a user-defined net. When released, the net returns to the resolved value.

```c
// user-defined datatype T
typedef struct {
    real field1;
    bit field2;
} T;

// user-defined resolution function Tsum
// with no access to present value
function automatic T Tsum(input T driver[]);
    Tsum.field1 = 0.0;
    foreach (driver[i])
        Tsum.field1 += driver[i].field1;
endfunction

nettype T wT;       // an unresolved nettype wT whose datatype is T

// a nettype 'wTsum' whose data_type is T and
// resolution function is Tsum
nettype T wTsum with Tsum;

// user-defined datatype TR
typedef real TR[5];

// an unresolved nettype 'wTR' whose data_type
// is an array of real
nettype TR wTR;
```
The following example shows how to use a combination of a parameterized class definition with class static methods to parameterize the data type of a user-defined nettype.

```plaintext
class NetBase #(parameter p = 1);
    typedef struct {
        real r;
        bit[p-1:0] other_data;
    } NetType;

    static function NetType Tsum(input NetType driver[]);
        Tsum.r = 0.0;
        foreach (driver[i])
            Tsum.r += driver[i].r;
    endfunction
endclass

typedef NetBase#(2) MyNetType;
nettype MyNetType::NetType narrowTsum with MyNetType::Tsum;

typedef NetBase#(32) MyNetType;
nettype MyNetType::NetType wideTsum with MyNetType::Tsum;

narrowTsum net1; // other_data is 2 bits wide
wideTsum net1;   // other_data is 32 bits wide
```

**In Annex, A.2.1.3 Type declarations, ADD,**

```
net_type_declaration ::= 
    nettype data_type net_type_identifier 
        [with [package_scope|class_scope] tf_identifier];
```

**In Table B.1 – Reserved keywords, ADD:**

```
nettype
```
In 6.7, 10.3 and A.2.1.3, CHANGE:

```
net_declaration ::= net_type [drive_strength | charge_strength] [vectored | scalared] data_type_or_implicit [delay3] list_of_net_decl_assignments;
```

TO:

```
net_declaration ::= net_type [drive_strength | charge_strength] [vectored | scalared] data_type_or_implicit [delay3] list_of_net_decl_assignments;
| net_type_identifier [ #delay_value | # (mintypmax_expression)] list_of_net_decl_assignments;
```
At the end of 6.7 Net declaration, ADD,

6.7.1 User-defined Net Declaration

A user-defined net (or atomic net) is a net whose nettype allows users to describe more general abstract values for a wire. A net declared with that nettype uses the datatype and any associated resolution function for that nettype.

```c
// an unresolved nettype wT whose datatype is T
nettype T wT;

// a nettype 'wTsum' whose datatype is T and
// resolution function is Tsum
nettype T Tsum with wTsum;

// a net of unresolved, user-defined type wT
wT w1;

// an array of user-defined nets of unresolved type wT
wT w2[8];

// a net of resolved, user-defined type wTsum
wTsum w3;

// an array of user-defined nets of resolved type wTsum
wTsum w4[8];

// user-defined datatype TR
typedef real TR[5];

// an unresolved nettype 'wTR' whose data_type
// is an array of real
nettype TR wTR;

// a net whose data_type is an array of real
wTR w5;

// an array of user-defined nets whose
// datatype is array of real
wTR w6[8];
```
6.7.2 Initialization of user-defined nets

The resolution function for any resolved atomic net shall be activated at time zero at least once. This activation occurs even for atomic nets with no drivers or value changes on drivers at time zero. Since the actual evaluation of the resolution function is subject to scheduling non-determinism, no assumptions can be made regarding the state of driven values during the guaranteed call; the call may precede or follow any time zero driver changes. After the initial call resolution function evaluation is scheduled based on changes to the drivers.

The initial value of an atomic net shall be set before any initial or always procedures are started and before the activation of the guaranteed time zero resolution call. The default initialization value for a user-defined net shall be the default value defined by the datatype. For an atomic net whose datatype is a struct type, any initialization expressions within the struct shall be applied.

At the end of 7.2.2, CHANGE:

The initial assignment expression within a data type shall be ignored when using a data type to declare a net (see 6.7).

TO:

The initial assignment expression within a data type shall be used for atomic nets (see 6.6.7 and 6.7.1) but shall be ignored when using a data type to declare a non-atomic net (see 6.7).
At the end of 6.10, CHANGE:

See 22.8 for a discussion of control of the type for implicitly declared nets with the `default_nettype compiler directive.

TO:

It shall not be legal to implicitly declare a user defined net.

See 22.8 for a discussion of control of the type for implicitly declared nets with the `default_nettype compiler directive.

In 10.3.2, CHANGE:

Nets can be driven by multiple continuous assignments or by a mixture of primitive outputs, module outputs, and continuous assignments. Variables can only be driven by one continuous assignment or by one primitive output or module output. It shall be an error for a variable driven by a continuous assignment or output to have an initializer in the declaration or any procedural assignment. See also 6.5.

TO:

Nets can be driven by multiple continuous assignments or by a mixture of primitive outputs, module outputs, and continuous assignments. Variables can only be driven by one continuous assignment or by one primitive output or module output. It shall be an error for a variable driven by a continuous assignment or output to have an initializer in the declaration or any procedural assignment. See also 6.5.

The left-hand side of a continuous assignment to a net with a user defined nettype shall not contain any indexing or select operations into the datatype of the nettype. Thus, a continuous assignment to a user defined nettype is not able to drive part of a net with a user defined nettype; the entire nettype value must be driven.
At the end of 10.3.3, CHANGE:

If the left-hand references a vector net, then up to three delays can be applied. The following rules determine which delay controls the assignment:
- If the right-hand side makes a transition from nonzero to zero, then the falling delay shall be used.
- If the right-hand side makes a transition to z, then the turn-off delay shall be used.
- For all other cases, the rising delay shall be used.

TO:

If the left-hand references a vector net, then up to three delays can be applied. The following rules determine which delay controls the assignment:
- If the right-hand side makes a transition from nonzero to zero, then the falling delay shall be used.
- If the right-hand side makes a transition to z, then the turn-off delay shall be used.
- For all other cases, the rising delay shall be used.

If the left hand references a user defined net or an array of user defined nets then only a single delay can be applied. The specific delay is used when any change occurs to the value of the net.
Note to the reader:
There are three variations of 23.3.3 included. The committee is still discussing which approach is more appropriate. The wordings are all included for discussion purposes. There was general consensus that either variant A or B is preferred.

[Variant A] In 23.3.3, ADD:

An internal atomic net for a port of mode output or input shall only be connected to an external variable or atomic net whose data type matches the data type of the port. Even if both the internal and external are atomic nets, the nets shall not be merged; the connection shall be treated as a continuous assignment through the port.

An internal atomic net for a port of mode inout shall only be connected to an external atomic net whose data type matches the data type of the port. In the case of an inout user defined net connection, the internal and external nets shall be merged into a single simulated net.

[Variant B] In 23.3.3, ADD:

An internal atomic net for a port of any mode shall only be connected to an external atomic net whose data type matches the data type of the port. The internal and external nets shall be merged into a single simulated net.

[Variant C] In 23.3.3, ADD:

An internal atomic net for a port of mode output or input shall only be connected to an external variable or atomic net whose data type matches the data type of the port. If both the internal and external are atomic nets, the internal and external nets shall be merged into a single simulated net. If the either the internal or external is not an atomic net, the connection shall be treated as a continuous assignment through the port.

An internal atomic net for a port of mode inout shall only be connected to an external atomic net whose data type matches the data type of the port. The internal and external nets shall be merged into a single simulated net.
In 23.3.3.3, CHANGE CLAUSE TITLE:

23.3.3.3 Port connection rules for nets

TO:

23.3.3.3 Port connection rules for built-in nets

In 28.12, immediately before 28.12.1, ADD:

User defined nets shall not have strength levels. Combining signal values for user defined nets shall follow the rules in 23.12 (Resolution of user defined nets)