Mantis 3398

SV-DC User defined types on nets

Alternative V1

Change notes for the reader:

Introduction

Verilog and hence SystemVerilog currently has one “physical” type that is used for modeling logic signals on wires. Verilog-AMS has that type and also uses “continuous” analog signals, which can be in multiple domains or “disciplines” (electrical, fluid dynamics, mechanical etc.), and adds methods for handling both when connected on the same net. The aim of the SV-DC committee is to fill in the spectrum between the pure analog behavior and the simple logic discrete modeling by adding support for user-defined types, and a generalization of the methods used in Verilog-AMS to handle cases where multiple types are used on a single net.

Simulation Objects

Physical Nets

A physical net is a construct in a hardware description language used to model actual hardware connections, usually a single electrical wire. In the analog simulation domain physical nets are collapsed to a single node in simulation where Kirchoff's Current Law (KCL) applies. All drivers and receivers attached to a physical net have to be of the same discipline, e.g. all electrical.

Abstract Nets

An abstract net is used in non-physical representations of a design. A design description can use abstract nets or physical nets with behavioral logic descriptions (e.g. RTL) as input to tools like synthesis which will output lower level models (closer to transistor level) and actual wiring (physical nets).

Processes

A process in an HDL is the code that represents hardware behavior at either abstract levels or as accurate low-level models.

Drivers and Receivers

Processes do not interact directly with nets. The interface for contributing a value to a nets is a “driver”, the interface for getting the value from the net is the “receiver”. The type of a net is usually determined by the types of drivers and receivers attached to it. Drivers and receivers are not explicitly declared in Verilog.

Receivers in Verilog-AMS are implemented as analog-to-digital converters which convert the voltage on a node to the logical representation expected by a digital process. Information about threshold voltage is stored in the disciplines, and may differ across receivers.

Kernel Net Objects

Certain net-types like "wire-or" have additional objects added for simulation – weak pull-ups, pull-downs or hold-ups.

Primitives

Verilog has a variety of built-in models for primitive switches, these generally work by relaying a strength-reduced signal from one side of the device to the other under control of a third input.

Existing Net Behavior

Verilog & Verilog-AMS

As mentioned above Verilog has one net-type, which for this document is called "strength-logic" and it is used for all nets, drivers and receivers. Other net types used in Verilog are essentially degenerate cases of that type, with restricted range. The type embodies three orthogonal concepts:

• Value	The logic level to be communicated
• Strength	The driver strength
• Certainty	How well determined is the value

For this document this is referred to as a 3-D net-type. One of the degenerate types is "4-state" logic where the bit (single wire) values are represented by 1,0,X or Z, where those correspond to the 3-D net-type as follows:

• Z	Strength is zero
• X	Certainty is zero
• 0/1	Logic level is low or high

Driver resolution - which is the method used to determine the net value when multiple drivers are attached – is simple for strength-logic:

1. determine which drivers have the highest strength and ignore the others, the net has that strength
2. the certainty of the net is the lowest certainty of those drivers
3. the net takes the value of those drivers unless they do not match, in which case then the certainty of the net is zero (and value indeterminate)

This method of resolution is commutative, so drivers can be resolved in groups, or you can consider the net as hierarchical object and resolve locally and propagate results up the hierarchy resolving at each level (see VHDL below). So the behavior is functionally the same whether you consider a net “collapsed” or not.

The above method does not require understanding how to resolve multiple value components, so there is no value-resolution function required.

The above method of resolution is error prone, a large number of weak drivers may outweigh a single strong driver in reality, but the algorithm ignores that. In Verilog-AMS the goal is higher accuracy, so each driver is converted individually to an analog contribution[1] and the the net is collapsed so that KCL applies and an analog solver can be used to determine the correct net[2] voltage.

VHDL

In VHDL all (digital) net-types are user-defined and resolution is performed hierarchically. Unlike Verilog-AMS analog and digital drivers cannot be mixed on a net, so for most purposes the VHDL modeling methodology should be considered as “abstract”. Explicit resolution functions are required in VHDL because the concepts of value, strength and certainty are conflated into enumerated types that the simulator kernel does not necessarily understand.

Proposal

Methodology

The existing methods for handling driver, receiver and net-types can be extended to work with new types by taking the VHDL approach of moving their definition out of the simulator into user-space as a header file that can be included (or assumed), in a way that other net-types can be derived from the existing behavior. This means defining the strength-logic net-type in terms of a template class, and also it's degenerate forms (e.g. “logic”). Separate header files would be provided for the template and the strength-logic definitions.

If a user wishes to use an alternative net-type definition for strength-logic they can do so by using a different header file. This is useful if you need to work around the errors in the existing approach without resorting to Verilog-AMS.

By default the resolution for derived types is the same as for the base template which is the algorithm described above: highest strength driver(s) value, with zero certainty on conflict. An explicit resolution function can be added to override that behavior, but it is defined externally to the net-type class since it's requirements may vary with design usage, and the return type of a resolution function may differ from the type of the drivers being resolved in order to provide greater accuracy. Resolution functions need not be pure, access to global data may be required to find information extra information about

1 Verilog-AMS parlance for an analog driver

2 The net is a single node in nodal-analysis

the drivers and the context of the resolution. If a resolution function is defined within a net-type it is assumed to be lossless[3] (rather than lossy, as is the default), e.g. the summing of currents.

Independently declared resolution functions are also considered as being general purpose conversion functions, and can be reused with a single element array as input for converting resolved net values to receiver types. Such functions can be marked as lossy to indicate that accuracy is lost in the conversion. E.g. converting a real-valued net representing a voltage to a logic value is a lossy conversion.

A user may declare a net-type as "abstract" in which case the net will not be collapsed across ports, and type conversions may be applied at the port[4]. Any net with drivers that are not using the default resolution scheme and physical net-types will be collapsed across ports. The default behavior is to collapse nets as that is the more accurate approach.

Net-type declarations for ports are only required in modules which have drivers and receivers. Physical net-types declarations used in contexts where no drivers or receivers are present are considered advisory and are ignored with respect to simulation semantics.

User defined net-types propagate through primitives the same way that strength-logic does if the user-defined type uses the same strength definition.

Net-types which are assignment compatible, i.e. values, strengths and certainties can be directly copied from the friend, are indicated by the use of the "extends" keyword. The compiler may issue warnings if the conversion appears to lose data or be otherwise incompatible.

Methods can be added to a net-type for evaluating posedge and negedge, noting that the edge determination is local to the receiver, not a function of the resolved net. If a net-type does not have the methods but it is in an extends statement in another net-type that does , then the other net-type's methods can be used, that allows degenerate types to defer to more accurate types for common functionality.

Some net-types are asymmetric with respect to driver strength (wor, wand), for those a class method (strength_limit) is used to modify the strength according to the value and certainty. When a driver is modified by the user, the method is called before the driver value is propagated into resolution.

For net-types requiring kernel objects an unamed module definition can be added to the class definition prefixed by the keyword "kernel", which will be instantiated once in the top module of the net for the final resolved type of the net.

Assignment

Verilog has two driver assignment mechanisms: blocking and non-blocking. Non-blocking assignments to members of an aggregate type in a net-type are propagated immediately. Assignments using blocking assigns are coalesced and processed once at the alloted time.

3 No accuracy is lost in performing the resolution

4 VHDL style

Assignments using the normal “4-state” logic representations are allowed for user-defined net-types. A Z-bit is interpreted as assigning 0 (Hi) to the strength member, and an X-bit as assigning 0 to the certainty. 0/1-bit values are assigned directly to the value if the type extends strength-logic using the default strength and certainty. If the type does not extend strength-logic then a suitable resolver/conversion function will be used if one is available, otherwise an error is reported.

Syntax

Standard Net-types

The net-type template class header, and supporting types:

```
typedef enum {false=0,true=1}            logic_value;
typedef enum {Hi=0,Sm,Me,We,La,Pu,St,Su} net_strength;
typedef enum {Hi=0,St}                   bool_strength;
typedef enum {unknown=0,known=1}         signal_certainty;

class nettype #(type value_t,
                type strength_t,
                type certainty_t);
      value_t      value;
      strength_t   strength;
      certainty_t  certainty;
endclass
```

The standard types header:

```
class logic extends
              nettype #(logic_value,
                        bool_strength    default St,
                        signal_certainty default known)
      // posedge,negedge from strength_logic
endclass


class strength_logic extends
                        nettype #(logic_value,
                                  net_strength     default St,
                                  signal_certainty default known)
      function int posedge(strength_logic_base next,
                           strength_logic last);
```

```
        function int negedge(strength_logic_base next,
                             strength_logic last);
        extends logic; // net_strength includes bool_strength
    endclass
    class wor extends strength_logic;
        kernel module (inout net);
            pulldown(net);
        endmodule
        function net_strength strength_limit();
    endclass
```

Resolution functions are special conversion functions, they take an array of drivers of one net-type and return a net-type which can be the same net-type or another net-type. Resolver functions can be called in the same way as any other function. The optional keyword “lossless” can be added before the routine name (always “resolver”) to indicate the conversion does not lose accuracy. The built-in strength-logic resolution function would have the following prototype:

```
    Function strength_logic resolver(input strength_logic drivers[]);
```

User-Defined Net-types

The Verilog-AMS wreal type can be implemented as:

```
    typedef nettype #(real,
                      bool_strength,
                      signal_certainty) wreal;
```

By default a wreal net with that definition will behave as you would expect for a voltage, if you want current-summing behavior you would add a specific resolution function:

```
    class summed_real extends wreal;
        function summed_real resolver(
                            input summed_real drivers[]);
    endclass
```

Nets driven by multiple types of driver are resolved by using resolution functions to convert the disparate net-types to a common net-type, with preference being given to lossless conversions for arrays of drivers, then lossless conversions of individual drivers followed by lossy conversions. The functions required for a mixed-net are determined at the end of elaboration and after any back-annotation. If no resolution scheme can be determined an appropriate error message will be issued, and simulation is not possible. If lossly conversions are used in a mixed-net resolution a warning will be issued.

A typical case requiring mixed nets is the use of normal Verilog logic modules with additional parasitic capacitors. Parasitic capacitors can be described with a real-valued "capacitance " net-type driver, and logic drivers can be converted to their Thevenin equivalent for resolution. This would require code similar to the following:

```
class thevenin_node;
      real voltage;     // voltage at last time
      time last;        //
      real capacitance; // node capacitance
      real current;     // summed driver current
      real resistance;  // parallel resitance
      kernel module (inout net);
            // voltage evaluation code here
      endmodule
endclass
class thevenin extends nettype#(thevenin_node,void,void);
      function thevenin resolver(input thevenin drivers[]);
enclass
class cap extends nettype#(real,void,void);
      function cap resolver(input cap drivers[]);
endclass
function thevenin lossless resolver (
                              input strength_logic drivers[]);
function thevenin lossless resolver (
                              input cap drivers[]);
// down conversion for receivers
function logic resolver (input thevenin drivers[]);
```

The "cap" net-type includes a (implicitly) lossless resolver, which can be used to sum all the capacitances on the net. The "cap" net-type is not jointly resolvable with strength_logic, but both can be converted to thevenin, so the "cap"drivers will be resolved to a single thevenin driver with only the capacitance value set, and the logic driver(s) will be converted to a thevenin driver with only current and resistance set. The two thevenin drivers are then resolved into one.

Supporting Functionality

The kernel modules added to manage a net may need to scan the drivers of the net and differentiate between those and any belonging to the kernel module. These are taken from Verilog-AMS:

```
int $driver_count(<signal_name>)
<net-type> $driver_state(<signal_name>,<driver_index>)
```

Verilog-AMS uses separate functions for strength and value, but since the net-type contains both only one routine is required.

An additional function indicates that a driver belongs to the calling context:

```
int $driver_self(<signal_name>,<driver_index>)
```

The return is zero for drivers not owned by the calling context, non-zero otherwise.