2016 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION UNITED STATES

SystemVerilog-AMS The Future of Analog/Mixed-Signal Modeling

Martin Vlach – Mentor Graphics

Scott Little – Intel

2016 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION UNITED STATES

Agenda

- Tutorial introduction
- Committee introduction
- Analog simulation concepts
- Requirements and broad concerns
- Object model, `nodetype`
- Connectivity
- Power aware connectivity
- Configurations
- Next steps

2016 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION UNITED STATES

Agenda

- **Tutorial introduction**
- Committee introduction
- Analog simulation concepts
- Requirements and broad concerns
- Object model, `nodetype`
- Connectivity
- Power aware connectivity
- Configurations
- Next steps

Target Audience

- Majority digital verification engineers
  - Most of them with current interest in MS verification
- Minority, if any, analog designers
- Vendor engineering/marketing folks
- Minority with knowledge of Verilog-AMS

Tutorial Objectives

- Let those interested know about what is happening
- Expose those with digital background to important considerations coming from the analog world
  - In particular those that affect SV-DC
- Minor objective: Expose those with analog background to important considerations coming from the digital world
  - In particular those that affect changes to V-AMS

2016 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION UNITED STATES

Key Takeaways

- MS verification is moving to digital-centric techniques, hence SV-DC importance
- Behavioral continuous time Verilog-AMS is not going away, hence SV-DC is not sufficient
- Verilog-AMS (based on Verilog 2005) is dead, hence need to move to SV-AMS
- Verilog-A subset for compact modeling will be preserved
- This will be IEEE work
  - IEEE PAR does not yet exist
  - 1800.1 has been reserved for SV-AMS

2016 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION UNITED STATES

Agenda

- Tutorial introduction
- **Committee introduction**
- Analog simulation concepts
- Requirements and broad concerns
- Object model, `nodetype`
- Connectivity
- Power aware connectivity
- Configurations
- Next steps

2016 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION UNITED STATES

The Committee

- The current committee was formed as a sub-committee of Verilog-AMS under Accellera
- We started meeting around April 2014
- It is composed of representatives from both Verilog AMS and SystemVerilog
- The cross-pollination of ideas from those two communities is essential

Committee Goals

- Merge Verilog-AMS and SV rejected early on
  - Verilog is dead (2005 last rev)
  - Verilog-AMS beyond 2.4 will not be approved by Accellera Board
- Work towards a point standard to IEEE 1800 (SV) that will extend SV to SV-AMS (just like VHDL is extended to VHDL-AMS)
  - Everything in 1800 is valid
  - AMS is a proper superset
- Prepare a white paper that covers all major considerations
  - This will be the basis for the IEEE 1800.1 SV-AMS standard.
- Prepare recommendations to the SV-DC (Discrete real modeling committee) to extend their work so that it is compatible with the anticipated SV-AMS

2016 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION UNITED STATES

Agenda

- Tutorial introduction
- Committee introduction
- **Analog simulation concepts**
- Requirements and broad concerns
- Object model, `nodetype`
- Connectivity
- Power aware connectivity
- Configurations
- Next steps

Conservative System Modeling

- General concept in multi-physics models
- Explained here using electrical terminology as it is the most important and familiar to this audience
- Both voltages and currents are essential and contribute to the model
- Lumped (not distributed) models
- Kirchhoff conservation laws
  - KCL - conservation of charge - currents flowing into a node sum to 0
  - KVL - conservation of energy - voltage drops along a loop sum to 0
- Analog models of a device describe how voltages and currents are related
  - static - relationships independent of time, Ohm's law, V=I*R
  - dynamic - changes over time, differential equations, I = dQ/dt

Conservative System Modeling

- Continuous time
- Differential Algebraic Equations (DAEs)
- Consequences
  - Everything depends on everything else
  - The kind of reasoning that an event causes another event is not useful
  - Digital is often concurrent; analog is always simultaneous
  - Digital is discrete, like algebra; analog is continuous, like calculus

Verilog-AMS and SPICE

- The A part of Verilog-AMS is targeted for implementation in SPICE simulators
- All solutions are approximations
  - Numerical algorithms for solving
    - differential equations (integration algorithms)
    - nonlinear equations (most commonly Newton-Raphson)
  - Accuracy vs. simulation time tradeoffs are always required
- SPICE handles responses (waveforms) that are continuous in time
  - No discontinuous changes, unlike digital simulation where all signal changes are discontinuous
  - Responses are computed at time samples, but the distance between samples can be made arbitrarily small to increase accuracy.
- Requires models where the dependency of current vs. voltage and charge vs. voltage is continuous and has continuous first derivative
- The algorithms may tolerate violations of these rules, but that will always introduce potential for error, non-convergence, increased simulation time, and problems in general

Verilog-A

- Historically is the precursor to Verilog-AMS
- Is now a defined, standardized, proper subset of Verilog-AMS
- Is essential to CMC (Si2 Compact Model Coalition, aka Compact Model Council) for compact device modeling
- Is used in PDKs

Key Points About Analog Simulation

- Time is a continuous value
- Waveforms are continuous in time
- At a given time, once a numerical solution is obtained, it cannot change (no such thing as delta cycles in SPICE algorithms)

2016 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION UNITED STATES

Agenda

- Tutorial introduction
- Committee introduction
- Analog simulation concepts
- **Requirements and broad concerns**
- Object model, `nodetype`
- Connectivity
- Power aware connectivity
- Configurations
- Next steps

Miles High View of Requirements

- Backward compatibility with text of Verilog-A device models
  - The detailed syntax of declarations in disciplines.vams need not (and will not) be backward compatible
  - Verilog-A is a clearly defined normative subset of SV-AMS
- Backward compatibility with Verilog-AMS is a goal but not a requirement
- Postpone handling of concepts that have never been considered in either V-AMS or SV
- Proper superset of SystemVerilog IEEE 1800

The Big Concerns

- Those concepts that were judged to be new to either the analog or digital community, difficult, or potentially controversial
- There are many details that will need to be taken care of, and may be hard to close on, but are not controversial
- Object kind "node" in SV-AMS (continuous domain)
- Use of the SV User Defined Nettypes to implement and extend `wreal` net of Verilog-AMS
- Use of SV `interconnect` for structure
- The ability to connect unlike signal representations
  - e.g. `electrical/logic/wreal` in Verilog-AMS, UDN in SV-AMS
- Supply-aware API for use in converting logic to/from voltage
  - Well-defined layer above UPF
  - Rich enough to handle non-UPF power/supply descriptions
    - Left for vendors to implement on top of legacy proprietary methods

The Big Concerns (cont.)

Rejected concepts from Verilog-AMS

- Use of `wire` as a structural connection
  - replaced by SV `interconnect`
- Use of hierarchical insertion of connect modules between adjacent net segments
  - replaced by insertion of adapters between abstractions
- Use of discrete disciplines and connectrules
  - replaced by adapter configurations and supply-aware API

Guiding Principles

These principles have been established for the work product of the committee in the White paper

- When conflict in terminology, concepts, meaning, syntax, semantics, etc. exists between the Verilog-AMS Language Reference Manual and the IEEE 1800 Standard for SystemVerilog, the usage from IEEE 1800 is retained whenever possible
- When conflict between Verilog-AMS LRM and IEEE 1800 is such that the AMS usage is prevalent in the EDA community, and the IEEE 1800 can be changed easily, the change is recommended
- The proposed standard SV-AMS language should allow users to efficiently use SV-AMS for mixed-signal verification
- Proposed constructs should address both the DUT and the testbench requirements

Broad Areas of Concern

Number of stars reflect amount of discussion devoted to the concept

- Object Model (**)
- Data Types (**)
- Hierarchy (*****)
- Scoping
- Physical Constants (*)
- Handling of Time
- Scheduling Semantics (*)
- Analog Blocks
- Analog Expressions
- Signal access functions (*)

Broad Areas of Concern (cont.)

- Assignments, Operators
- Assertions
- Coverage
- Power-aware API (**)
- APIs
- SPICE Compatibility (*)
- Real literals
- Keywords (*)
- Math keywords and built-in functions (*)

2016 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION UNITED STATES

Agenda

- Tutorial introduction
- Committee introduction
- Analog simulation concepts
- Requirements and broad concerns
- **Object model, `nodetype`**
- Connectivity
- Power aware connectivity
- Configurations
- Next steps

Object Model

- Nets (as in SV)
- Variables (as in SV)
- Nodes (as in V-AMS)
- Branches (as in V-AMS)
- Continuous time (aka analog) variables (rationalized based on V-AMS)
  - Temporary continuous variables
  - Retained continuous variables
  - Initialized continuous variables

Data Types: nodetype

Explained by an example: Abbreviated contents of file “AmsPkg.sv”:

```
package AmsPkg;

// Current in amperes
nature Current;
    units = "A";
    access = I;
    idt_nature = Charge; // Definition of Charge omitted for brevity
    abstol = 1e-12;
endnature

// Potential in volts
nature Voltage;
    units = "V";
    access = V;
    idt_nature = Flux; // Definition of Flux omitted for brevity
    abstol = 1e-6;
endnature
```

Data Types: nodetype

(continued...)

```
// Conservative discipline
nodetype {
    potential Voltage,
    flow Current
} electrical;

// Signal flow disciplines
nodetype {
    potential Voltage
} voltage;

nodetype {
    flow Current
} current;

endpackage : AmsPkg
```

`nodetype`: rationale

- The addition of a new kind enables us to define rules for nodes independently of the existing SystemVerilog rules for nets and variables
- The `nodetype` name mirrors the existing SystemVerilog syntactic structure for nets and nettypes and gives SystemVerilog-AMS a consistent look and feel with existing SystemVerilog
- `nodetype` is not a SV `type` and cannot be used like a `type`
  - For example, a `nodetype` cannot be used as the `type` in a type parameter (P1800-2012 6.20.3). This is consistent with `nettype`

2016 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION UNITED STATES

Agenda

- Tutorial introduction
- Committee introduction
- Analog simulation concepts
- Requirements and broad concerns
- Object model, `nodetype`
- **Connectivity**
- Power aware connectivity
- Configurations
- Next steps

Concepts/Terminology

- **Signal**: (From IEEE 1800 standard Glossary *as extended*): An informal term, usually meaning either a variable, net, *or a node*. (Also sometimes “real life signal”).
- **Net, Var**: (SV LRM 6.2) A data object is a named entity that has a data value and a data type associated with it, such as a parameter, a variable, or a net.
- **Node**: A node is a named entity with an associated potential value (across variable in physics) or flow value (through variable in physics) to model signal flow analog abstractions, or both potential and flow to model conservative analog abstractions.
- **SPICE Node**: A named entity in a SPICE netlist. In concept it is equivalent to a node in SV-AMS, and in fact an analog engine should implement a node and SPICE node in the same way in the analog solver.
- **Simulated Net**: (SV LRM 23.3.3.7) The result of merging a dominating and dominated net into a single net, whose type is that of the dominating net. Contrast with **Collapsed Net** – the dominated net in such merging.
- **Simulated Node** (proposed): The result of merging a dominating and dominated node into a single node, whose nodetype is that of the dominating node. Contrast with **Collapsed Node**– the dominated node in such merging.
- **Port**: a means of structurally defining connections between two objects (net, node, var, structural connectivity object) that are in two different name spaces.

SPICE and the LRM

- SPICE description is an important piece of Analog and Mixed Signal descriptions and aspects of connecting SPICE and SV-AMS are part of the LRM
- It is undecided in the committee whether SPICE integration should be normative or informative
- The following aspects of incorporating SPICE into SV-AMS DUT and Testbench should be addressed in the SV-AMS LRM:
  - Instantiate a SPICE primitive and .subckt from SV-AMS
  - Ability to instantiate an SV-AMS module in a SPICE netlist
  - SPICE structural netlist
  - Associate a nodetype with SPICE primitive or subcircuit pin
  - .connect
  - .global (similar in concept to node in an SV package)

Representation of a Signal

- The representation of a real life signal in an HDL is uniquely identified by specifying its:
  - **Object Kind** (net, node, SPICE node). The modeling kinds are
    - Event-driven (net)
    - Conservative (node, SPICE node)
    - Signal flow (node)
  - **Object Nettype** or **Nodetype** (including for SPICE node).
  - **Resolution Function** (for net)
- Note: A datatype alone is not sufficient to determine a Signal Representation. Event-driven signal representation requires a nettype (which may be unresolved)
  - A var represents a value of a signal
- Note: The Verilog-AMS concept of “discrete discipline” is discarded from SV-AMS

Connectivity

- Concrete connectivity
  - Representation of the connecting object is explicitly specified
- Abstract connectivity
  - Representation is not known from the perspective of the instantiating module/subcircuit
    - In Verilog-AMS this is the structural `wire`
  - SV `interconnect`
    - In SV it is classified as a net, but in SV-AMS it also will include a node
  - SPICE structural node
    - Node in a subcircuit that is not connected to any SPICE primitive (in the subcircuit)
    - Port of a subcircuit not connected to any SPICE primitive (in the subcircuit)

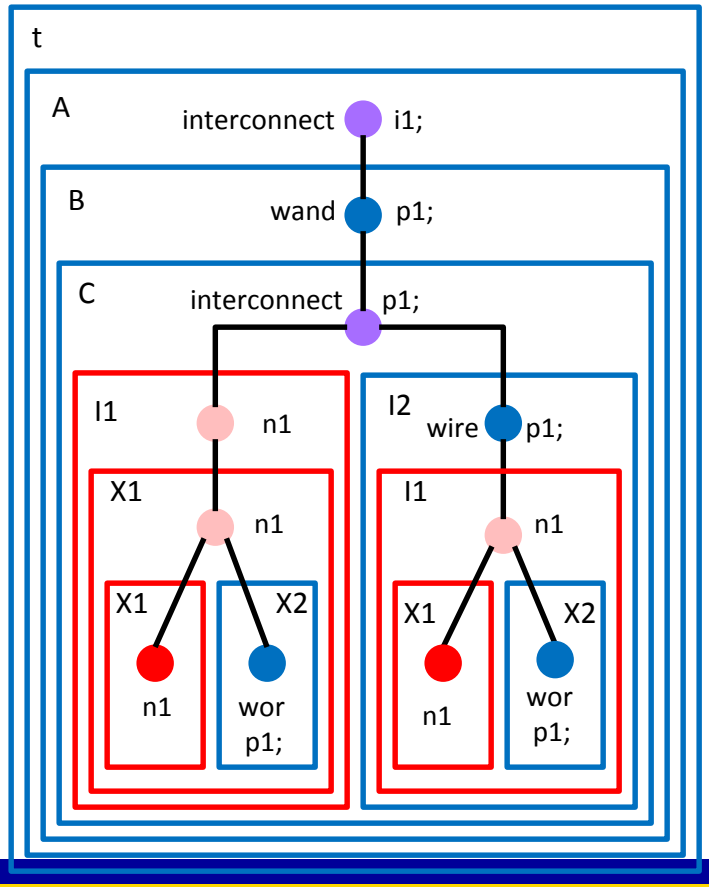
Simple Connectivity

SV

AMS/SPICE

| Port association

- interconnect
- SPICE structural
- logic
- electrical



$root.t.A.i1

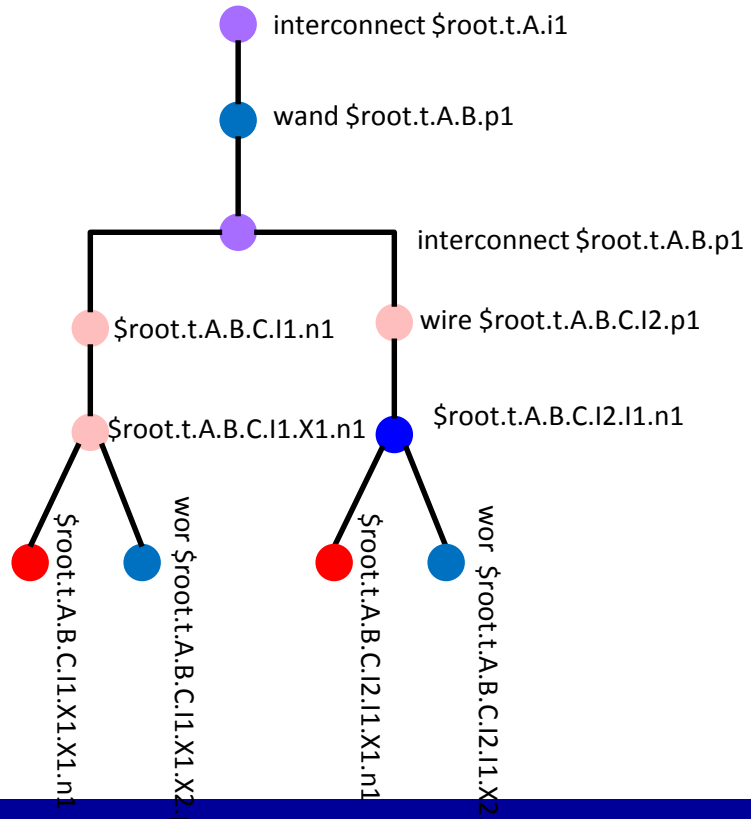
$root.t.A.B.p1

$root.t.A.B.p1

$root.t.A.B.C.I1.n1
$root.t.A.B.C.I2.p1

$root.t.A.B.C.I1.X1.n1
$root.t.A.B.C.I2/I1.n1

$root.t.A.B.C.I1.X1.X1.n1
$root.t.A.B.C.I1.X1.X2.p1
$root.t.A.B.C.I2.I1.X1.n1
$root.t.A.B.C.I2.I1.X2.p1



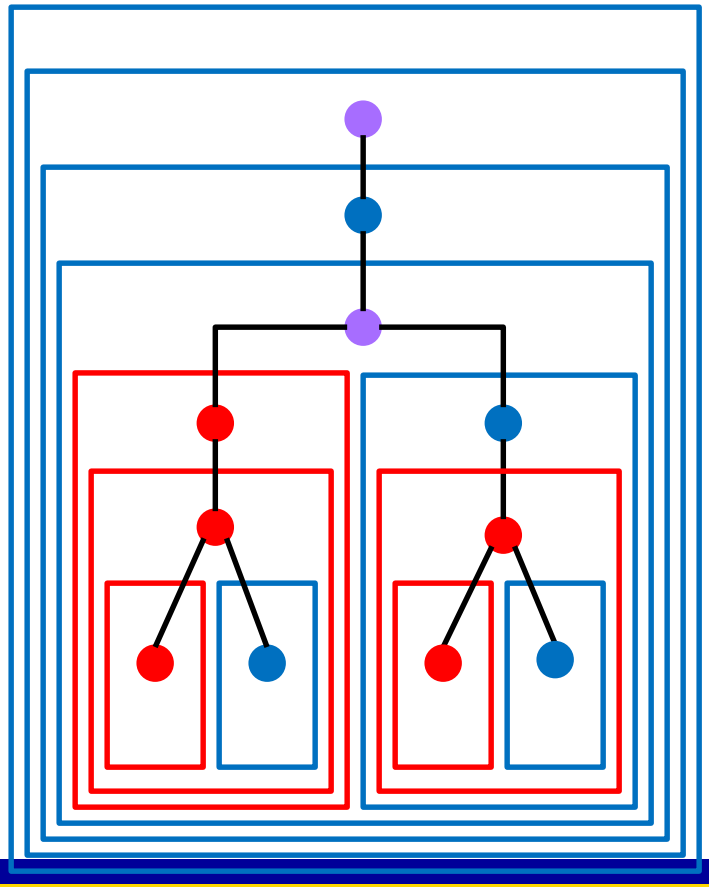
Hierarchical Conversion in Verilog-AMS

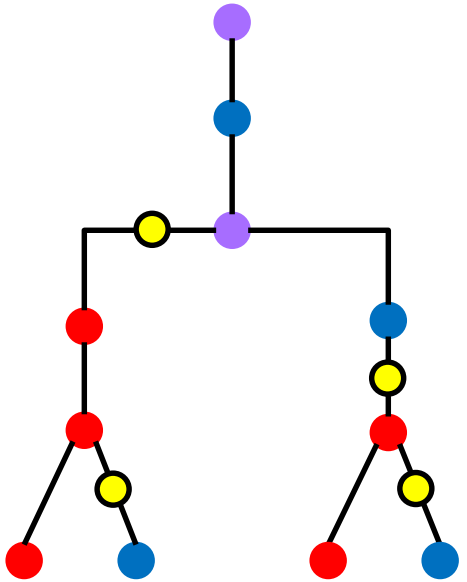
SV

AMS/SPICE

| Port association

- structural wire
- logic
- electrical
- connect module





Representation Conversion in SV-AMS

SV

AMS/SPICE

Port association

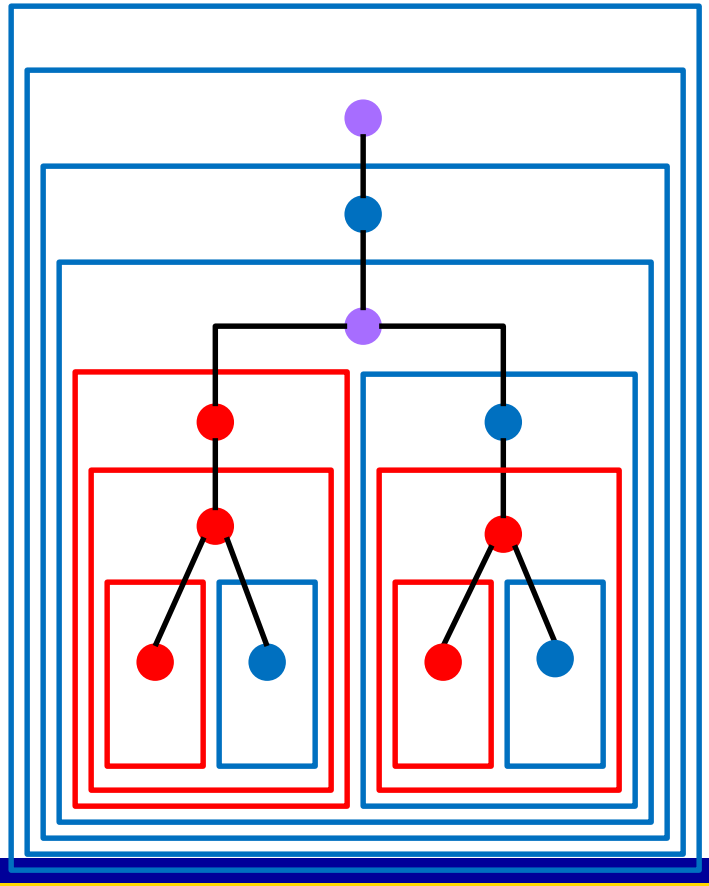
Implied connection

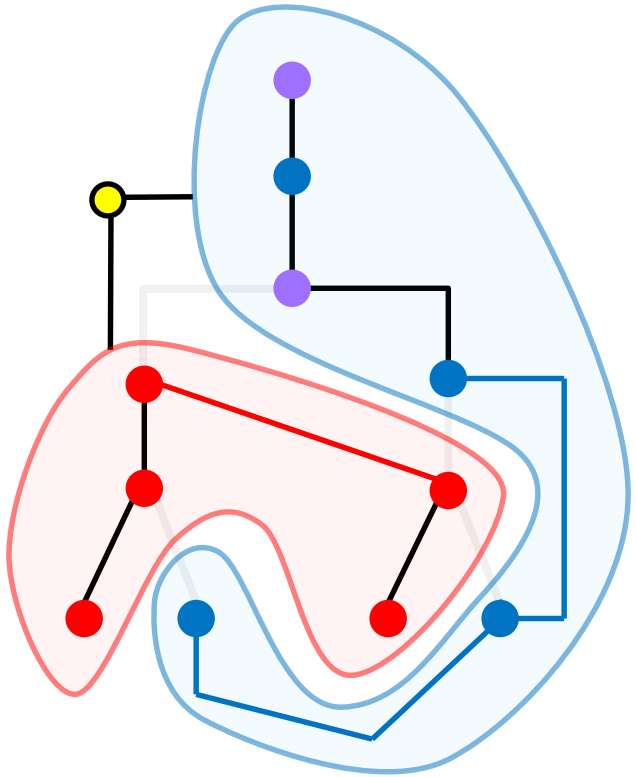
interconnect

logic

electrical

adapter





Complex Connectivity

HDL

SPICE

Any

Port association

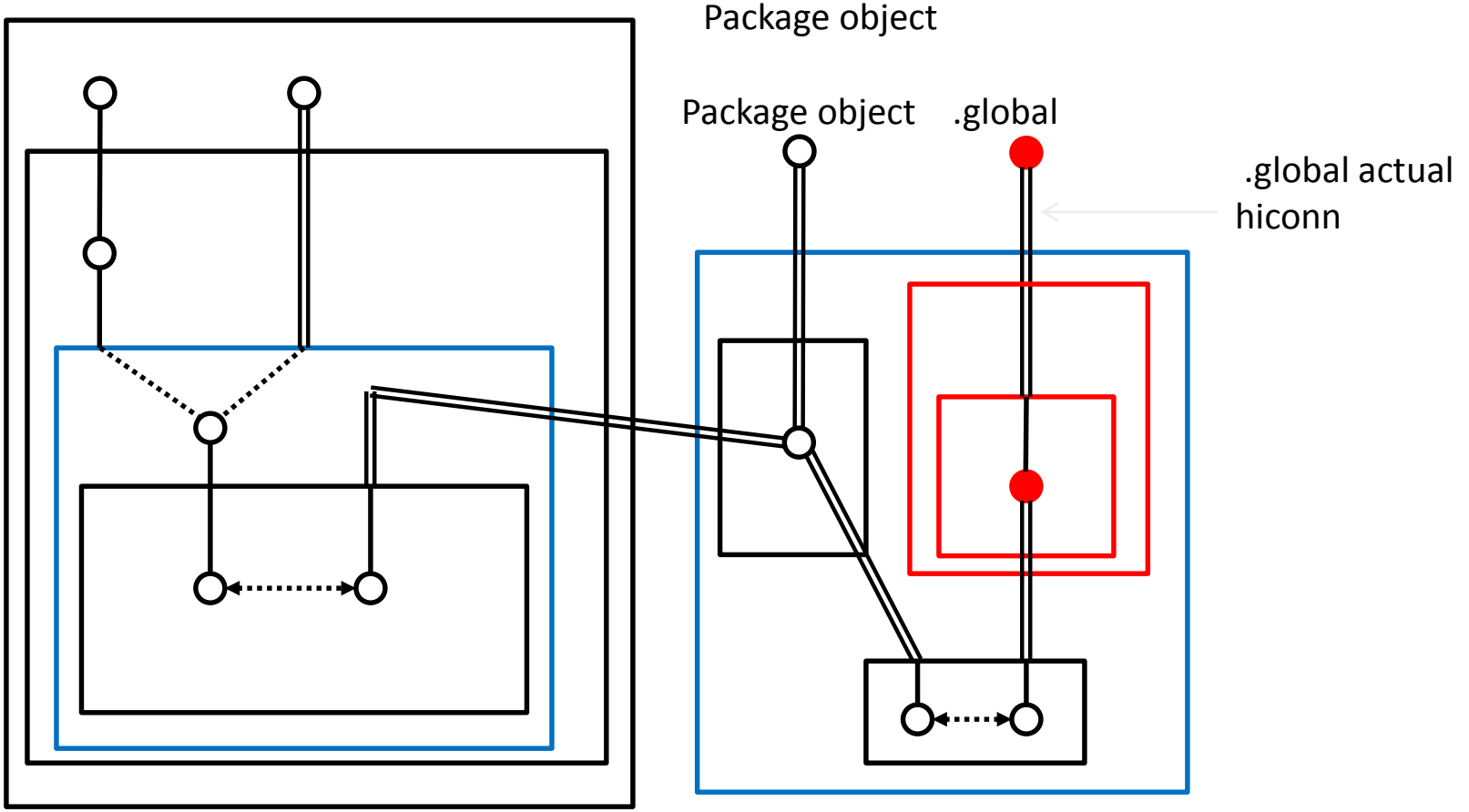
Port aliasing

.connect,
Analog alias

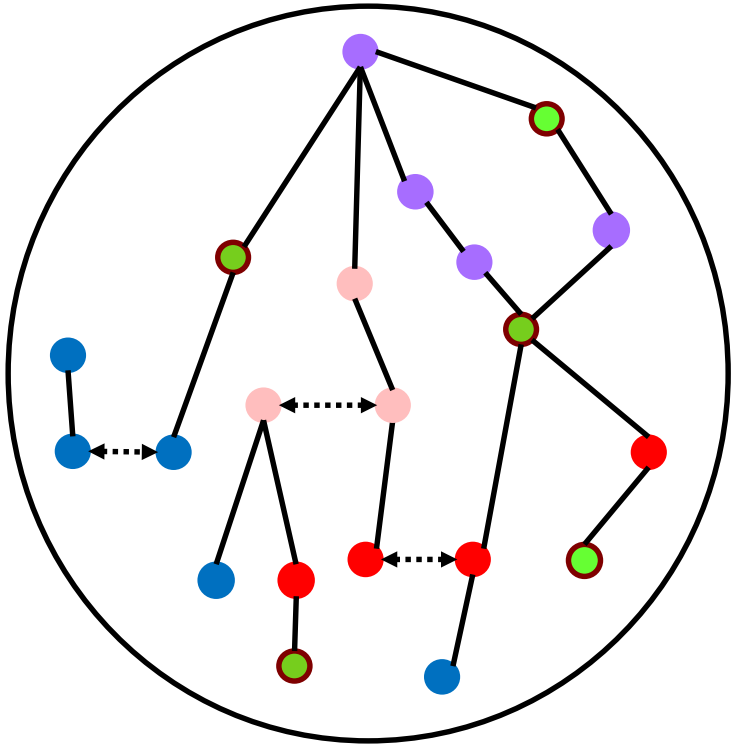
Href port association
.global
Package object

electrical

any



The Connectivity Graph



Port association, .global
Port aliasing

.connect,

- interconnect
- SPICE structural
- logic
- electrical
- UDN0 (real/max)
- UDN1 (real/sum)
- UDN2 (not real)

Note: The graph is not a tree. There can be multiple tops, and there can be cycles. There is no “root” or “highest node” in the connectivity graph.

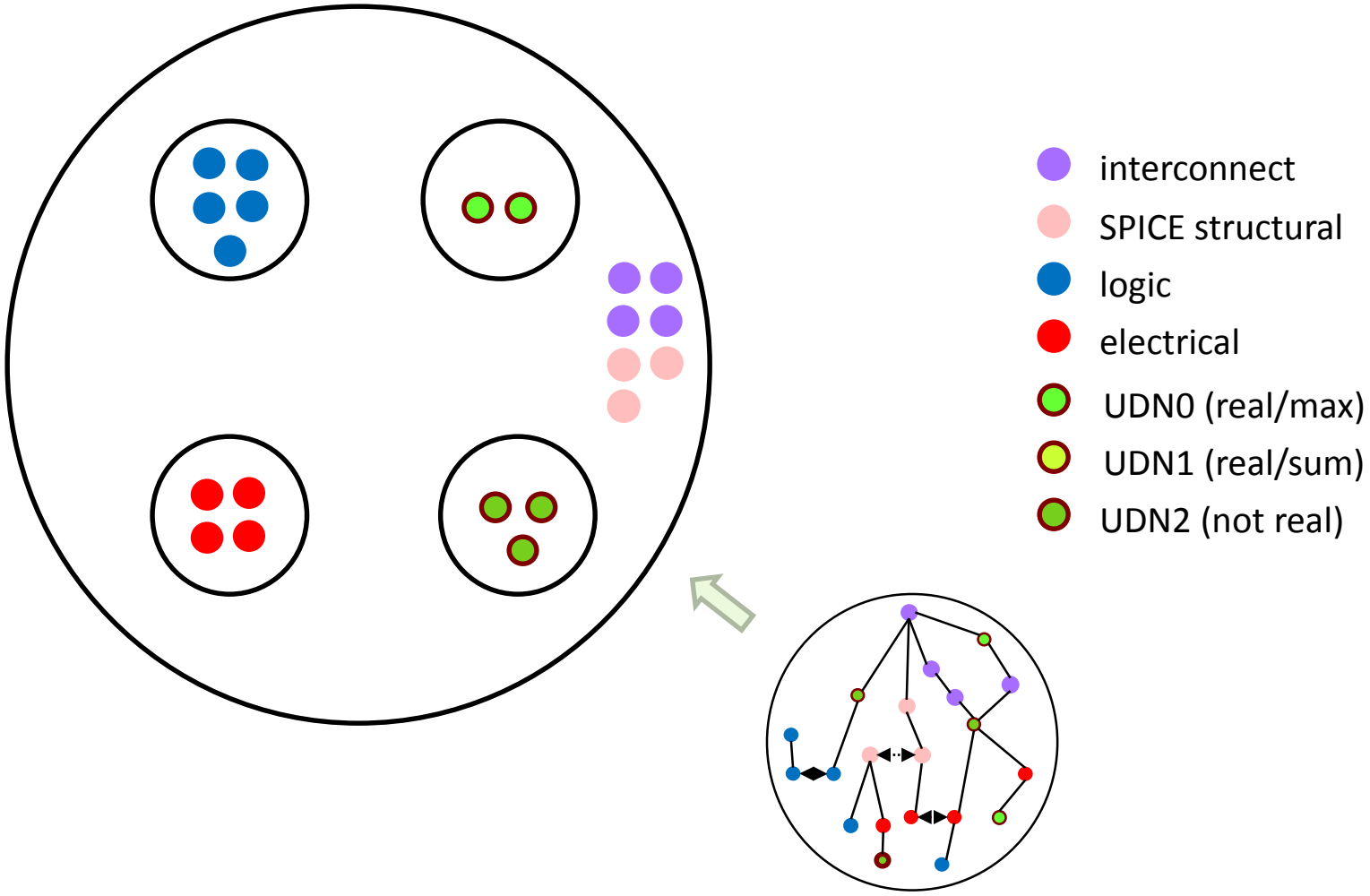
Supernet

- **Definition**: **Supernet** is the set of objects formed by the transitive closure of objects of Object Kind net, node, and SPICE node through
  - Port associations
  - Verilog net aliases
  - Verilog port aliases
  - SPICE `.connect` commands
- `interconnect` is an untyped net or a node; SPICE structural node is a (untyped) SPICE node, and thus a supernet also includes all abstract connectivity objects
- `tran` does not participate in the transitive closure
- A `var` is not part of a supernet, in keeping with SV notion of a net and net collapsing. Colloquially, “var breaks a signal” in SV

Supernet - Discussion

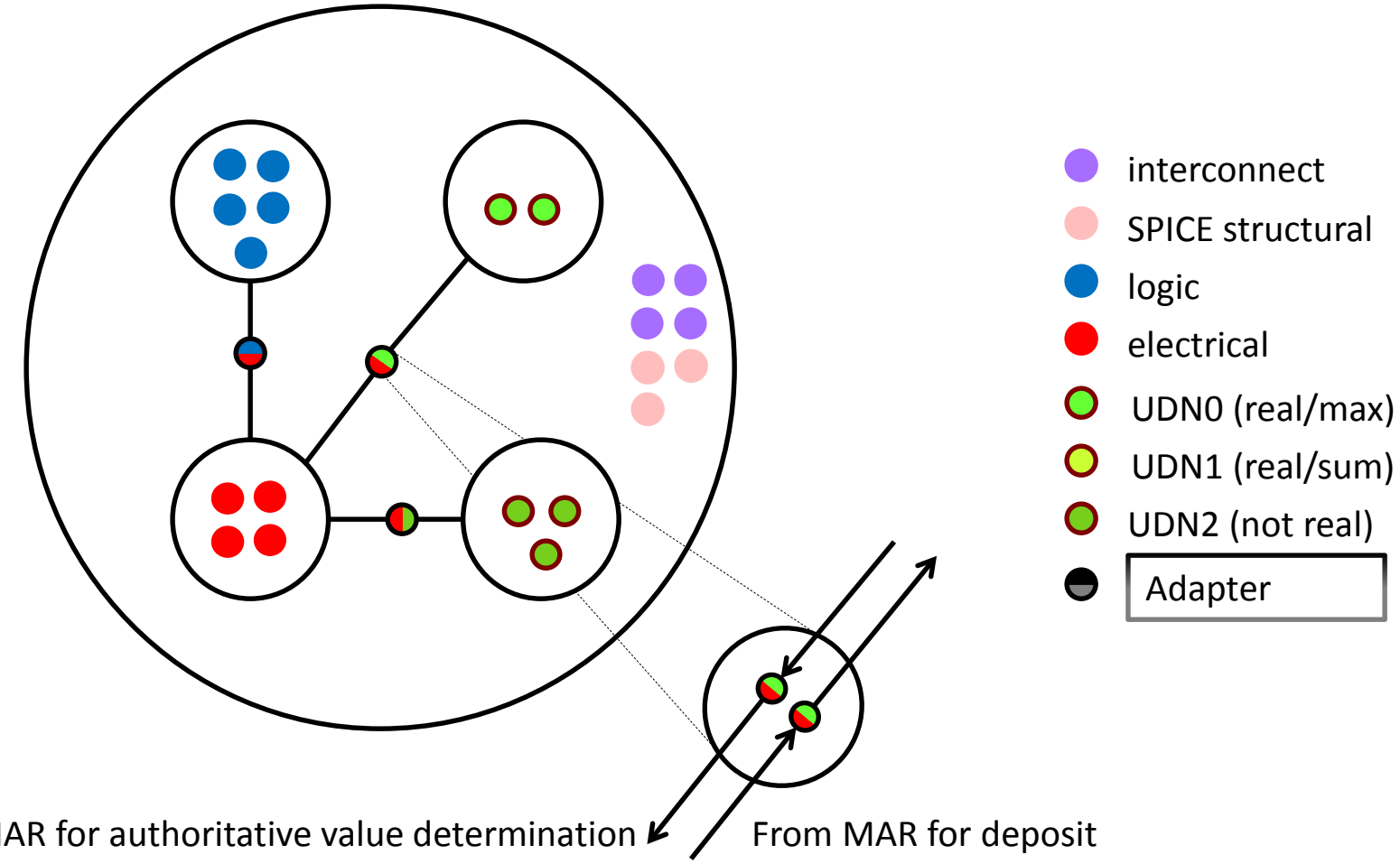
- *Guiding principle*: All objects in a supernet that have the **same nettype/nodetype** shall have the **same value** at all times, regardless of how they are connected
  - When considering electrical voltage, a way to visualize a supernet is as (an approximation of) an equipotential region
- Port associations in an instantiation or bind of a module (or program or interface) may contain
  - Locally declared objects
  - Hierarchical references (Hrefs, aka OOMRs)
  - References to objects imported from an SV package
  - SPICE `.global` nodes (details left to SPICE implementers)
- Objects in a supernet may have compatible or incompatible signal representations, or be representation-less (abstract connectivity); these are subject to
  - Existing SV rules in case of built-in net types
  - **Representation conversion for all other cases**

Merging


interconnect
SPICE structural
logic
electrical
UDN0 (real/max)
UDN1 (real/sum)
UDN2 (not real)

Adapter Insertion with MAR

MAR: MAster Representation



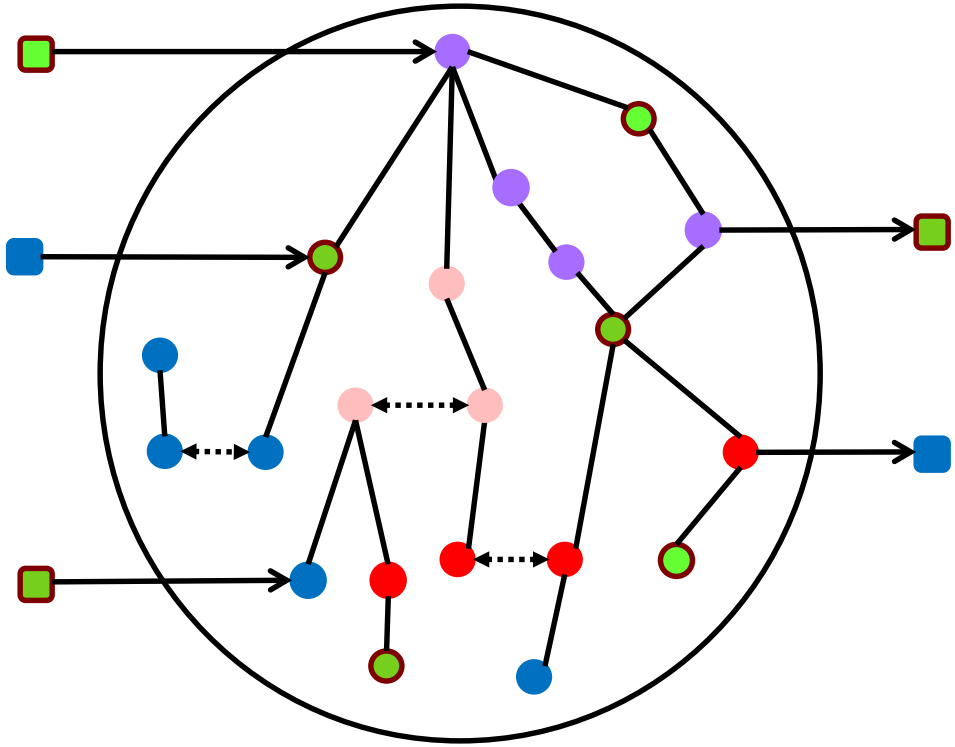
Representation Conversion: Adapters

- Representation conversion is achieved by *Adapters*.
- Committee considered several adapter mechanisms
  - R-to-R (Representation-to-Representation) – Preferred
  - Many-R – Rejected (late)
  - Chained – Rejected (early)
- Event-driven resolution (within each nettype) can be done
  - Using the adapter code or MAR explicitly. Most accurate, preferred.
  - Using the nettype resolution function, with the resolved value going into the adapter. The simulator calls the resolution function. Loss of accuracy is likely.
  - Implementations may “pre-resolve” subsets of the drivers, thus “collapsing” several drivers into one value that is presented to the adapter. This could be done for various reasons such as performance optimization or partial elaboration. The details are not part of the LRM.

R-to-R Conversion

- R-to-R conversion uses the concept of MAster Representation (MAR). Conceptually
  - All representations are converted to the MAR using a unidirectional adapter
  - The determination of an authoritative value is performed within the MAR
  - The MAR value is converted to each contributing representation using another unidirectional adapter, as the final “resolved” value
- Notes:
  - MAR specification is per supernet (not global)
  - R-to-R conversion for a supernet may use a MAR that does not exist in that supernet
  - R-to-R conversion always requires a pair of unidirectional adapters for each of the representations in the supernet that are not the MAR
  - It can be used to implement Verilog-AMS style of connect modules **provided** that the high impedance state of a node can be determined by the analog engine
  - The specification of which representation is the MAR (within any given supernet) must be provided to the elaborator by the user, it is not prescribed by the LRM

Supernet and vars



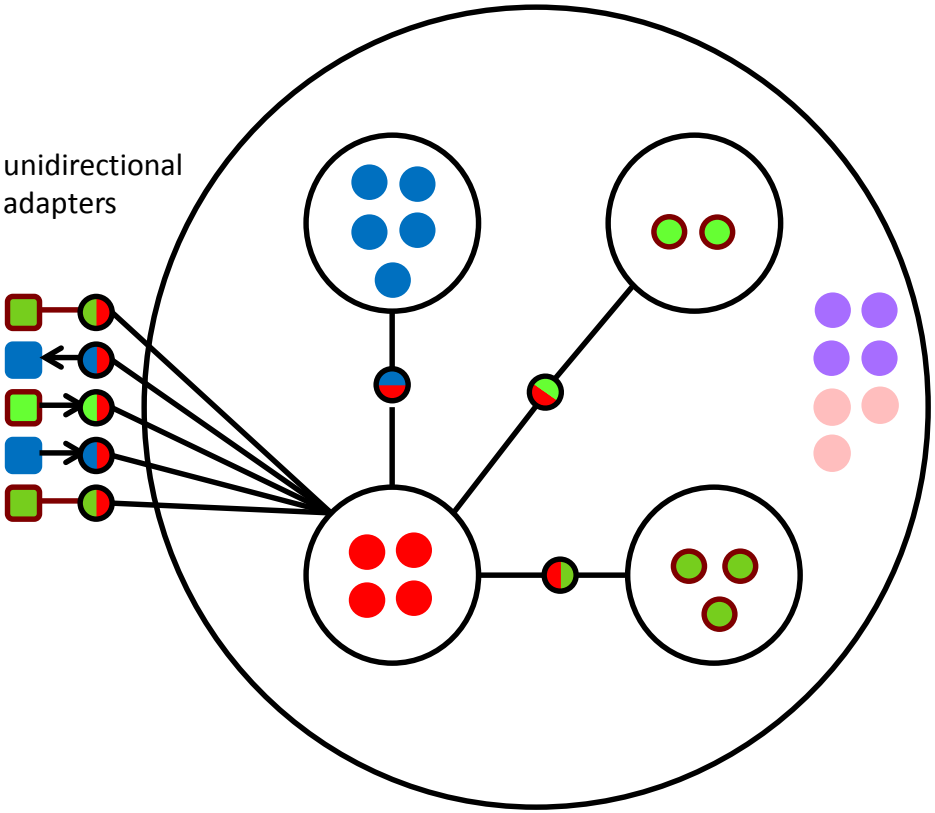
Port association (local or hierarchical – Href)
.global
Port aliasing

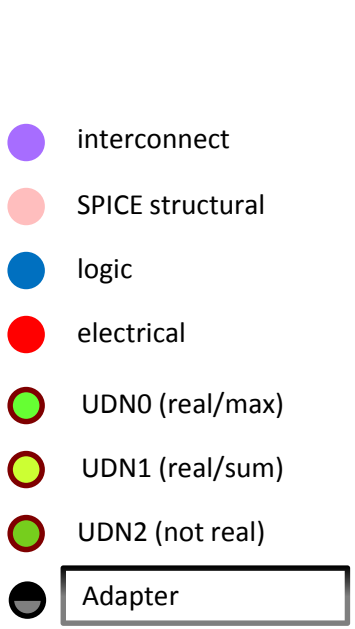
.connect,

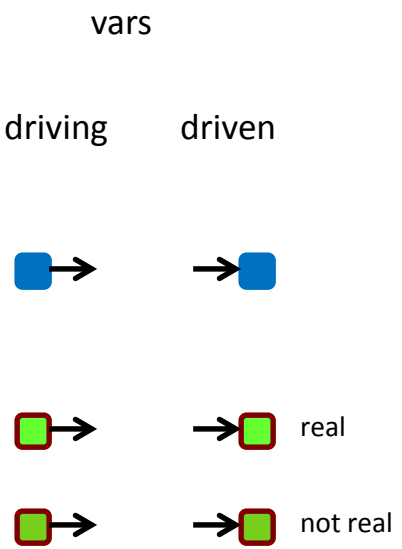
vars

	driving	driven	
interconnect			
SPICE structural			
logic	→	→	
electrical			
UDN0 (real/max)	→	→	real
UDN1 (real/sum)			
UDN2 (not real)	→	→	not real

Adapter with MAR and vars







Picking Adapter for a var

- There may be many nettypes that have the same type and differ only in resolution function: real/max and real/sum
- Adapter choice is based on nettypes, not types
- Picking which adapter to use for which var (which nettype to associate with the var's type) may not be unique in a supernet (it is in the earlier example)
- Proposed rules
  - If a supernet contains exactly one nettype that has the same type as the var, the adapter for that nettype is used for converting from/to the var's value by the adapter
    - Note that the var may be connected to any of the objects in the supernet, including objects of other kinds and types, or type-less (abstract connectivity)
  - Otherwise, such ambiguity must be resolved by a configuration

Tri State Logic: Recap

- Uses the concept of a high impedance state usually referred to as “Z”
- Is represented by 1’bz in the logic type
- In analog circuits a node with high impedance implements tri state logic.
- Problem: the node voltage always has some value in the analog circuit, which would normally be converted to 0,1, or X on the logic side
  - High impedance node’s voltage drifts over time depending on 2nd order effects
  - The knowledge that a node is in the Z state is important for logic simulation
- If we need to send Z to the event-driven side, we need help from the analog solver – Z analog state cannot be determined from the node voltage
  - Modern SPICE simulators can provide this information
- Not (analog) driving a node (net) is represented by (digitally) driving the Z value

Impedance System Function

- Boolean value: is node in high impedance state?
  - `$hiz(node[,threshold])`
- Monitored Analog Event: when impedance state changes
  - `@hiz[node[,threshold[,enable]])`
- Value: what is the impedance of a node?
  - `$impedance(node)`

The ams_pa package

- Power aware functions needed by adapter code to handle conversion to/from logic levels are available in the standard SV package `ams_pa`
- `supply1Value(<arg>), supply0Value(<arg>)`
  - Returns the real value, in Volts, of the supply associated with `<arg>`, (if state is `FULL_ON` or `PARTIAL_ON`, or 0.0 if the power domain state is `OFF` or `UNDETERMINED`
  - `<arg>` is
    - The name of a port (usually `logic`, but not required)
    - Indexed name of a port (for individual driver access if needed: `linp[i]`)
- `supply1Change(<arg>[,vdelta[,tdelta]]), supply0Change(...)`
  - Returns an event when a supply for any of the drivers (or for the specified driver), changes state, or its value changes subject to the V/time window
    - Default values for the window to be decided
- `supply1State(<arg>), supply0State(<arg>)`
  - Returns the UPF state of the supply associated with `<arg>`

Properties of Adapters

- Each adapter has exactly two ports, which may be
  - `input nettype1, output nettype2` (w/ `nettype1` != `nettype2`)
  - `inout nodetype, output nettype`
  - `input nettype, inout nodetype`
  - For each `input nettype1, output nettype2`, there must be a corresponding `input nettype2, output nettype1`
  - For each `input nettype, inout nodetype`, there must be a corresponding `inout nodetype, output nettype`
  - The ordering of ports as "`input,output`" or "`output,input`" or "`input,inout`" or "`inout,input`" *etc*. is not material.

Electrical-to-Logic Adapters

```
adapter electricalToLogic
#( parameter real SUPPLYVAL = 1.0 )
 (
   inout  electrical in,
   output logic out
 );
   always @(absdelta(V(in),1e-3)) begin
     if (V(in) > 0.5*SUPPLYVAL)
       out <= 1'b1;
     else
       out <= 1'b0;
   end
endadapter
```

Logic-to-Electrical Adapters

```
adapter logicToElectrical
#(parameter real SUPPLYVAL = 1.0) (
   input logic in,
   inout electrical out
 );
  real inVal = 0.0;
  always @(in) begin
    if (in === 1'b1) inVal = SUPPLYVAL;
    else inVal = 0.0;
  end
  analog begin
    V(out) <+ transition(inVal);
  end
endadapter
```

Electrical-to-Logic Adapters

```
//assumes that supply0 is always 0v
adapter electricalToLogic (
  inout  electrical in, output logic out
);
  always @(absdelta(V(in),1e-6),
           $supply1Change(out),
           $hizChange(in)) begin
    if ($hiz(in))
      out <= 1'bz;
    else if (V(in) > 0.5*$supply1Value(out))
      out <= 1'b1;
    else
      out <= 1'b0;
  end
endadapter
```

Logic-to-Electrical Adapters

```
adapter logic_to_electrical
#(parameter real rdrv = 25.0, real rz = 1e9, real rx = 1e3) (
  input logic in, inout electrical out
);
  real inLvl = 0.0;
  real rout = rz;
  always @(in, $supply1Change(in)) begin
    if (in === 1'b1) begin
      inLvl = 1.0;  rout = rdrv;
    end
    elsif (in === 1'b0) begin
      inLvl = 0.0;  rout = rdrv;
    end
    elsif (in === 1'bz) rout = rz;
    else begin
       inLvl = 0.5; rout = rx;
    end
  end
  analog begin
    V(out) <+ transition(inLvl*$supply1Value(in)) + I(out)*transition(rout);
  end
endadapter
```

nettype Definition

```
typedef struct {
   real v;
} rvoltT;

function automatic rvoltT
rvoltT_rf_avg(input rvoltT driver[]);
 //Implements an average of all drivers
endfunction

nettype rvoltT rvAvg with rvoltT_rf_avg;
```

R-to-R Adapters I

```
adapter rvAvgToElectrical (
   input rvAvg in,
   inout electrical out
);
   analog begin
      V(out) <+ transition(in.v);
   end
endadapter
```

R-to-R Adapters II

```
adapter electricalToRvAvg (
  inout electrical in,
  output rvAvg out
);
  initial begin
    out.v = V(in);
  end
  always @(absdelta(V(in), 1e-3))
    out.v = V(in);
endadapter
```

nettype Definition

```
typedef struct {
  real v;
  logic active;
} rvDriveT;

function automatic rvDriveT
rvDriveT_rf_avg(input rvDriveT driver[]);
  //Implements the average of active drivers
endfunction

nettype rvDriveT rvDrive with rvDriveT_rf_avg;
```

R-to-R Adapters III

```
adapter rvDrive_to_electrical
#(parameter real rdrv = 25.0, real rz = 1e9, real rx = 1e3) (
  input rvDrive in,
  inout electrical out
);
  real rout = rz;
  always @(in.active) begin
    if (in.active === 1'b1) rout = rdrv;
    elsif (in.active === 1'b0) rout = rdrv;
    elsif (in.active === 1'bz) rout = rz;
    else rout = rx;
  end
  analog begin
    V(out) <+ transition(in.v) + I(out)*transition(rout);
  end
endadapter
```

R-to-R Adapters IV

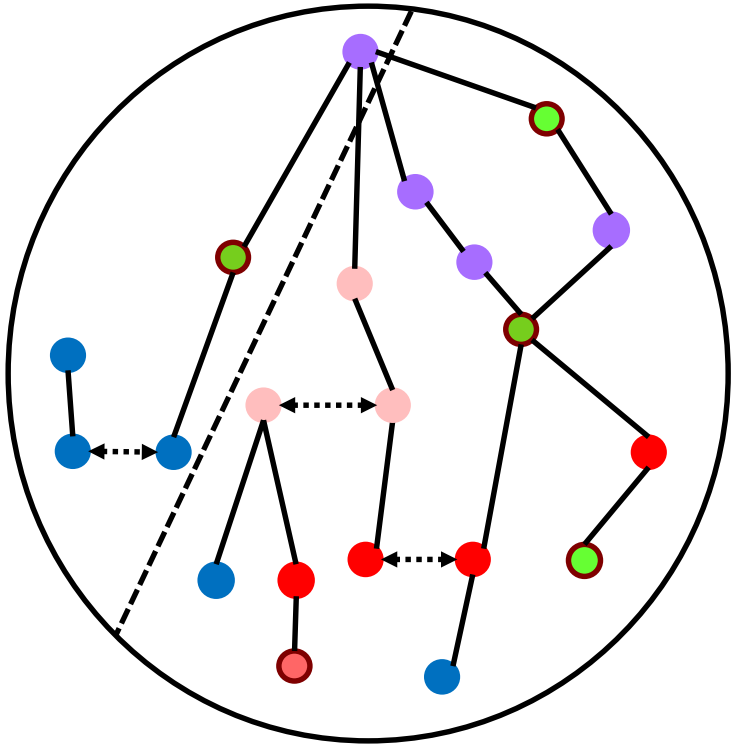
```
adapter electrical_to_rvoltD (
  inout  electrical in,
  output rvDrive out
);
  rvDriveT outVar = '{0.0,1'b1};
  assign out = outVar;
  always @(absdelta(V(in), 1e-3))
    outVar.v = V(in);
  always @($hizChange(in)) begin
    if($hiz(in)) outVar.active = 1'bz;
    else outVar.active = 1'b1;
  end
endadapter
```

2016 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION UNITED STATES

Agenda

- Tutorial introduction
- Committee introduction
- Analog simulation concepts
- Requirements and broad concerns
- Object model, `nodetype`
- Connectivity
- **Power aware connectivity**
- Configurations
- Next steps

The Connectivity Graph and UPF



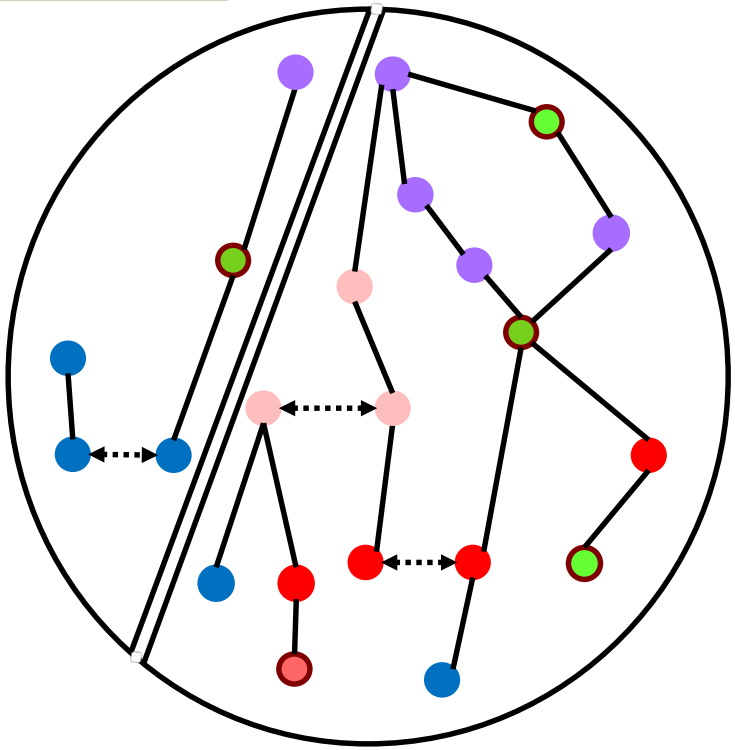
Port association (local or hierarchical – Href)
.global
Port aliasing

.connect,
Verilog net alias
Analog alias

interconnect
SPICE structural
logic
electrical
UDN0 (real/max)
UDN1 (real/sum)
UDN2 (not real)

Power domain separation
as specified by UPF

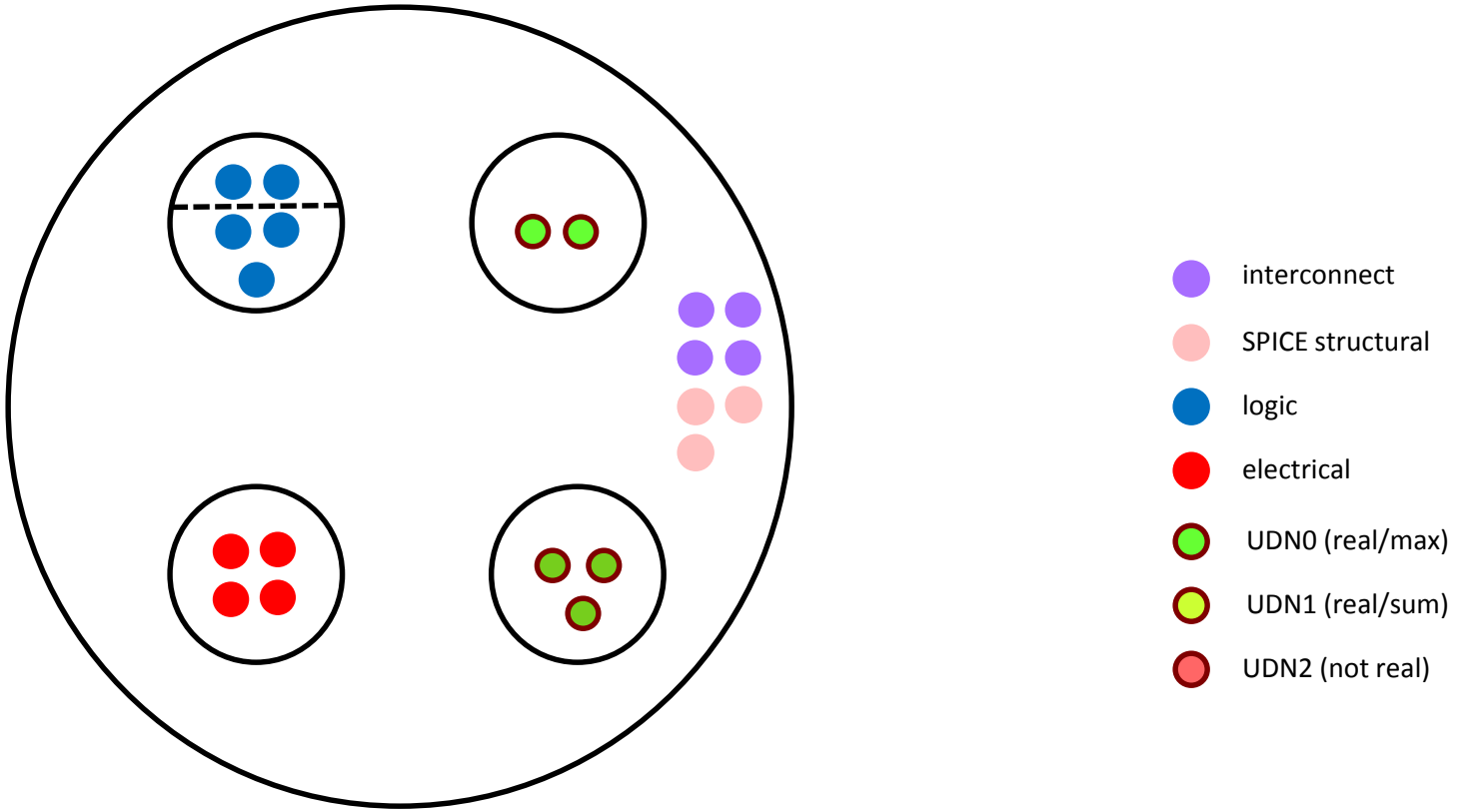
The Connectivity Graph and UPF



Application of UPF resulted in creation of two supernets, each handled independently.
No further issues.

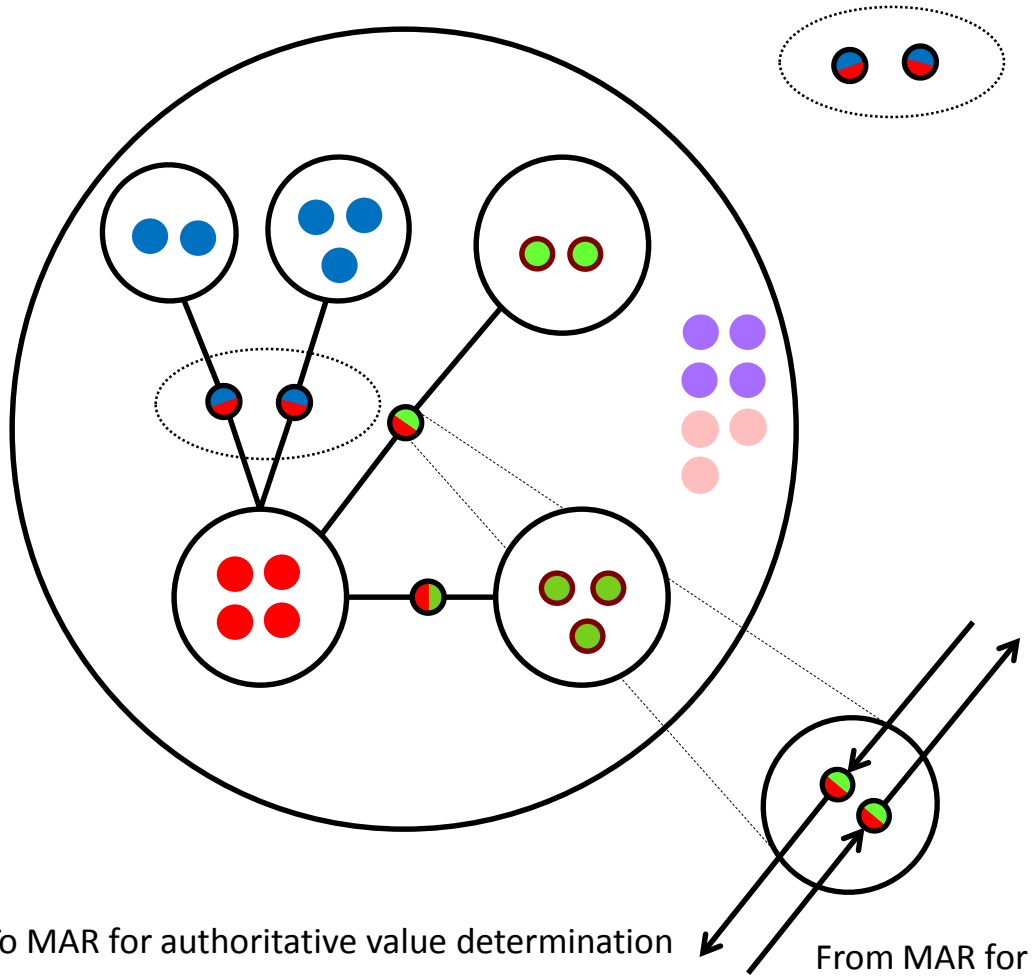
Power domain separation after **insertion of UPF elements**: supernet is divided

Merging and UPF



Power domain separation with
no insertion of UPF elements

Adapter Insertion with MAR and UPF



2016 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION UNITED STATES

Agenda

- Tutorial introduction
- Committee introduction
- Analog simulation concepts
- Requirements and broad concerns
- Object model, `nodetype`
- Connectivity
- Power aware connectivity
- **Configurations**
- Next steps

Definitions and Concepts

- ***Supernet*** is a collection of ***abstract connectivity*** objects (interconnect, SPICE structural node) and ***concrete objects*** (nodes and nets)
- ***n-type*** is a nodetype, nettype, or built-in net type
- ***MAR*** is the n-type that determines the authoritative value for objects in a supernet
- ***Resolved drivers adapter port*** for nettype X:
  - The adapter input port for X is scalar: `input X port`
  - If necessary, the simulator must first call the resolution function associated with X
- ***Unresolved drivers adapter port*** for nettype X:
  - The adapter input port for X is an array: `input X port[]`
  - The simulator does not call the resolution function of X before calling the adapter

Definitions and Concepts

- ***Adapter set***: Set of adapter declarations
- ***Parametrized adapter set***: An adapter set together with all parameter values to make it fully specified for use in determining the values of objects in a supernet
- ***Valid adapter set***: An adapter set that obeys all rules and restrictions on the n-types. The rules will be described later
- ***n-type family*** is a set of n-types. It may be defined:
  - By a supernet (the set of n-types of the concrete objects in the supernet)
  - By an adapter set (all the n-types of all the adapter ports in the set)
  - By specifying the n-type members of the set

Observations

- Vars are not members of a supernet
- A valid supernet must contain at least one n-type
- A valid supernet may contain many nettypes but at most one nodetype
- No adapter chaining is allowed
  - For every supernet there is one MAR with one set of to-and-back adapters
- The values of objects in a supernet are determined by one set of R-to-R adapters in cooperation with the MAR
- The MAR may, but need not, be one of the n-types in the supernet

Properties of Valid n-type Family

- n-type family cannot be empty
  - to be useful in further discussion
  - because an adapter set cannot be empty
  - because a supernet must have at least one concrete object (with an n-type)
- The obvious case for adapter insertion has size(n-type family) > 1
- Size 1 is an edge case which in the absence of time behavior is identical to a resolution function for nettype
  - For simplicity, it is disallowed, as it brings complexity without benefit, and in any case such delay would be better handled by an explicit model

Properties of the MAR

- The definition of which n-type in an n-type family is the MAR is required
- The simulator must know which n-type is the MAR, because the process of determining the adapted value is not symmetrical w.r.t. the MAR
- If a nodetype is present in a supernet
  - The nodetype will often be the MAR
  - The nodetype need not be the MAR - some other n-type (nettype) can be
- If the MAR is a nettype
  - The MAR *must have a resolution function*
  - The adapter MAR port must be scalar (resolved drivers port)

Properties of Adapter Sets

- One of the n-types (nettype or nodetype) must appear in all of the adapters in the set; it will become the MAR by default
- When size(n-type family) == 2, and
  - one n-type is a nodetype, then it becomes the MAR by default
    - the user may specify the nettype to be the MAR instead
  - both n-types are nettypes, then the user must specify which one is the MAR
- The adapter set must have an even number of adapters
- Any of the nettype input ports other than the MAR may be resolved drivers (scalar) or unresolved drivers (array) port
- The MAR port must be resolved drivers port (scalar)
  - If the MAR is a nettype, it must be a resolved nettype
- The nodetype (if present) must be scalar

Adapter Configuration

- Riffing on plain configuration from the SV LRM:
  An adapter configuration is simply an explicit set of rules to specify the exact set of adapters and their parameters to be used with each supernet in a design. The operation of selecting an adapter set for a supernet is referred to as binding an adapter set to a supernet.
- Adapter configuration must designate
  - The (sub) hierarchy ("the design") for which it applies
    - SPICE subcircuit may be specified
  - Where to find the adapters (libraries)
  - One or more adapter sets, each set with
    - The specification of the MAR (implicitly (default), or explicitly)
    - Parameterization
  - Additional filtering rules to pick one adapter set for each supernet in the design

Adapter Set Matching Rules

- Adapter set matches a supernet iff the adapter set n-type family (ASF) is a superset of the supernet instance n-type family (SIF).
  - Including when the ASF == SIF
- An adapter set with a nodetype never matches a supernet that has only nettypes
  - Don't force a nodetype (which would require analog solver) into a nettype-only (discrete time) situation
- Additional rules for selecting candidate adapter sets are described later
- After all of the rules on adapter set matching are applied, there may still be several adapter sets that match the supernet as candidates
  - Filtering rules can be applies
- Note: The user does not create these matching rules; they are part of the LRM. The use does create filtering rules.

N-types for Examples

```
nodetype {...} electrical;

nettype real real_nt with real_res_function;
nettype struct {...} custom with custom_res_function;
nettype struct {... ... ...} dmar with dmar_res_function;

// Abbreviations:
// e - electrical
// r - real_nt
// w - wire (not using "el" for logic since we are discussing
//      nettypes and built-in net types; "logic" would be a var
// c - custom
// dmar - "digital MAR" is an artificial (synthetic) nettype
//        that does not otherwise appear in the design.
//        It demonstrates the use of MAR to handle combinations of
//        r/w/c (i.e. not electrical)
```

Possible Configurations

```
// Possible supernet configurations that can appear in the
design
// with this set of n-types
// Size 2:
//    e,r
//    e,w
//    e,c
//    r,w
//    r,c
//    w,c
// Size 3:
//    e,r,w
//    e,r,c
//    e,w,c
//    r,w,c
// Size 4:
//    e,r,w,c
```

Basic Example

```
adapter configuration basicConfig;

design blockTop;

liblist lib1 lib2; // Where to find adapter definitions

adapter_set rwe r2e w2e e2r e2w;
adapter_set rw w2r r2w with real_nt;
adapter_set univ r2dmar w2dmar c2dmar dmar2r dmar2w dmar2c with dmar;

end adapter configuration

// Supernet        Matching Adapter Set
// Size 2:
//   e,r               rwe
//   e,w               rwe
//   e,c                -
//   r,w               rw, univ (but not rwe)
//   r,c               univ
//   w,c               univ
// Size 3:
//   e,r,w             rwe
//   e,r,c              -
//   e,w,c              -
//   r,w,c             univ
// Size 4:
//   e,r,w,c            -
```

Basic Example with Filtering

```
adapter configuration filteredConfig;

localparam v3_0=3.0;

design blockTop;

adapter_set rwe_30
    r2e #(.vsup(v3_0))
    w2e #(.vsup(v3_0))
    e2r #(.vsup(v3_0))
    e2w #(.vsup(v3_0));

adapter_set univ ...

instance blockTop.mux use rwe_30;
cell CPU use univ;

end adapter configuration
```

Filtering Examples

```
adapter configuration advancedFilteredConfig;

adapter_set set1 ...;
adapter_set set2 ...;
adapter_set set3 ...;

instance blockTop.radio
         for net1 use set1
         for net2 use set2
         for netX* use set3

cell lib1.CPU use set1
cell lib2.CPU use set2

cell opamp
         for netAna* use set3

instance blocktop_*.inst1 use set1
instance blocktop.*.inst1 use set1
instance blocktop...inst2 use set2

end adapter configuration
```

Processing a Configuration

- Precedence lowest to highest:
  - Library
  - Cell
  - Name (wildcarded)
  - Name (explicit)
  - Instance of module
  - Instance of name (wildcarded)
  - Instance of name (explicit)
  - Additional design needed:
    - lib, lib.cell, lib.cell.name, lib.name, cell.name
    - Similar for explicit instances
    - Wildcarding needed in the combinations?
    - Multiple hierarchies in one rule

Filtering Wildcards

- On names: Regular expression (RE) matching (details TBD)
- On hierarchies:
  - RE for a single hierarchy level
  - "..." to span zero or more hierarchy levels
  - All expansion is rooted at the `design` of the configuration
  - There is no going up (i.e. no ".." as in a directory structure)

2016 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION UNITED STATES

Next Steps

- Clean up the Whitepaper
- Transition the work done in Accellera to IEEE committees
  - SV-DC for handling discrete domain
  - New 1800.1 committee to handle the continuous domain
- References:
  - White paper: http://bit.ly/14ATQ8L
  - Accellera Committee site: http://workspace.accellera.org/apps/org/workgroup/sv-vams

2016 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION UNITED STATES

Thank You!