#	WebSite RefNum (flatten)	WebSite Ref Item _SubItem	LRM Section	Title	Description	Ranking 1 (HI) 2(MED) 3(LOW)	Approve (Y / N)	Comments
1		1	9.2.2	Truncating vs Rounding when converting Analog to Digital times	Truncating time on conversions will (on average) reduce simulation time vs. real time for the hardware, making the simulation optimistic. Also, if the user swap analog and digital components the timing is more likely to be off and lead to "false" errors in simulation results. Diagrams (Not included)			
2	2	2	8.9	Driver-receiver Segregation	Connect statements and the insertion of A/D & D/A converters is actually an extended form of 'signal resolution' - both VHDL and Verilog have mechanisms where all drivers of a signal have their values merged and the result appears as the value for that signal everywhere. The driver-receiver segregation section in the Verilog-AMS LRM as it stands appears inconsistent with this accepted approach and should be restated. [In general for a net that appears in multiple domains, the drivers in those domains need to be converted to the domain of highest accuracy, resolution is performed in the domain of highest accuracy (analog) and the result converted back to the lower accuracy domains.]			
3	3	3	8.8	A/D Convertor placement	Auto-inserted convertors should be on the child side of ports where conversion takes place as they are really associated with the driving and receiving processes in the child (not the parent), and because back-annotation may not work correctly otherwise. Also, if A/D convertors need to attach to a local power supply (say 'local.vdd' in Verilog) the search scope should start in the child. For 'merged' A/D convertors (where one converter serves multiple modules) the placement should be in the module that contains all the relevent processes in it's subhierarchy.			
4	4	4	8.10	Driver Type Function (ENH)	Proposal: We should add a "driver access" function for finding the type of a driver e.g.: driver_type_function ::= $driver_type (signal_name , driver_index); It would return an integer (defined in a header file) indicating the type of the driver. The user's connection module code can query it and decide how to handle updates on that driver. Possible header file values: // Driver Access Definitions: driver.h `define DRIVER_UNKNOWN 0 // Or-able bit flags `define DRIVER_DELAYED 1 `define DRIVER_GATE 2 `define DRIVER_BEHAVIORAL 4 `define DRIVER_SDF 8 `define DRIVER_TRANSPORT 16 `define DRIVER_RELIABLE(D) ((D) & (`DRIVER_DELAYED|`DRIVER_SDF)) `define DRIVER_UNRELIABLE(D) (((D) & `DRIVER_BEHAVIORAL) && ! ((D) & `DRIVER_SDF)) Note: Simulators can return '0' if unable to implement this functionality.			
5	5	5	9.1	A/D Synchronization	See LRM: 9.1 Analog simulation cycle The A/D synchronization algorithm is described at a kernel level rather than at the behavioral process level. I would prefer a description that is more general and applicable to multi-solver and multi-kernel simulators, e.g.: Step 1: Processes execute and schedules future values (and set d? /dt etc.) and a callback event (at acceptance, if necessary). Step 2: Simulator steps to global next event (analog or digital) and executes it. Step 3: Processes activated by event execute and reschedule future values and events. Step 4: Goto step 2 NB: Time is continous, 'digital time' doesn't really exist. The diagrams at the end of the link above confuse the issue by "hopping" around in time. (See "Time Continued") Disjoint analog blocks in a mixed-signal simulation need not use the same matrix solver and some analog blocks may not need a solver at all if they are entirely "Signal flow".			

6	6	6	AnnexE	External Module Definitions (ENH)	Requirement: Verilog-A[MS] is intended to replace the use of SPICE netlists. In the initial cut of the language the approach was taken that models (primitive analog modules) were implicit and that any modules used in a Verilog-A description that were not declared would be found in the simulator. This 'implicit declaration' approach has the problem that a design tool other than the actual simulator cannot tell if a design is complete. Modules which are required by a design but only supplied by a simulator need to be declared 'external' (as routines in C program modules are). Proposal: I propose extending the Verilog-A syntax by allowing the keyword 'extern' before a module declaration, and that such a module declaration can only contain port, parameter and variable declarations (i.e. no processes). I would also like to propose that the file including the standard (SPICE?) definitions is placed with the other simulation system include files accessible with: `include <models.h> Failure to locate a module definition would not be considered an error (but may cause a warning to be issued). The information about the class of the module and the supporting simulators should be handled by a standard attribute mech			
7	7	7	NA	Back-Annotation (ENH)	Compatibility: Verilog-AMS is expected to be a superset of Verilog. Verilog uses SDF for back-annotation, which is a methodology that does not require re-netlisting. For full compatibility Verilog-A[MS] also needs a mechanism that supports back-annotation without re-netlististing. Methodology: Top-down/bottom-up design usually involves designing functional blocks (modules) and then connecting them together to create larger functional blocks initially without interconnect. Those designs are then pushed through "place & route" to produce a "physical" design. The design after P & R has the same hierarchy as before, but the port connections are no longer simple connections but wires on Silicon. The working assumption with SDF was that wires can be treated as mostly capacitive - which was mostly true when it was invented. In "deep-submicron" circuits wiring is relatively much longer and suffers from relatively higher resistance and more crosstalk. A mechanism is required to back-annotate the actual circuit of the wiring between the modules to model it accurately.			
8	801	8_1.2	1.2	Mixed-signal language features	Access to analog signals from digital behavioral blocks, and vice versa. This is where the largest differences appear from earlier language levels. The features are defined further in Chapter 8 and other places.			
9	802	8_3.3	3.3	Genvars	Cleaned up from earlier versions (which had a generate statement). There are significant issues of scoping, etc, related to genvar. Verilog-2k has different mechanisms for this. This looks like a highly unportable feature.			
10	803	8_3.4.3	3.4.3	3.4.3.{2,3} Empty disciplines; undeclared nets	Description allows for netlists with uncommitted interconnect. The intention is clear although the explanatory text is highly ambiguous. Antrim AMS supports netlists using wire as an interconnect using psuedodisciplines.			
11	804	8_3.4.5	3.4.5	Ground declaration	Changed completely from previous usage (and totally incompatible with it, since ground now becomes a different kind of keyword). Need a migration plan for present users.			
12	805	8_3.5	3.5	Real net declarations	Changed (and relocated) from previous definition. This is an attempt to formalize $realtobits(), etc. Most of the semantics and syntax is missing from the definition. No rules provided for conversions with other net types. Expect Verilog-2k to conflict. This looks like a highly unportable feature.			
13	806	8_3.6	3.6	Default discipline	This applies to discrete disciplines only. Requires further definition (e.g. relationship with `reset_all).			
14	807	8_4.4.1	4.4.1	Restrictions on analog operators	See earlier comments on genvar. Also, this LRM introduces null arguments to these and other functions; this in turn introduces the need for various default values (which are not all provided by the LRM).			
15	808	8_4.4.7	4.4.7	Absolute delay operator	The LRM renames the previously named delay operator (which was a potential source of conflict with existing Verilog-D decks). Will need to support 'delay' as an alias for a migration period.			
16	809	8_4.5	4.5	Analysis dependent functions	Extended and better defined (with tables) in this LRM. Meaning of some cases (see also initial_step) is not clear for AC analysis, etc.			
17	810	8_6.7	6.7	Events	This LRM allows mixture of analog and digital events. This is a major new feature. It allows constructs like @(posedge clk or cross(V(1), 1)). Requires analog/digital synchronization semantics to be *much* better defined.			
18	811	8_6.7.4	6.7.4	Global events	Additional definition provided (including a table). The meaning of 'analysis point' is not clear.			
19	812	8_7.1.1	7.1.1	Top-level modules	Are multiple top-level modules that contain analog behavior allowed?			
20	813	8_7.3	7.3	Ports	[Not the right place for this]: need rules for vector versus scalar connections if the entity is reg rather than a wire type.			
21	814	8_7.3.3	7.3.3	Real valued ports	Require considerably more in the way of definition, especially how these interoperate with existing Verilog-D types, semantics, and syntax.			
22	815	8_8.2	8.2	8.2.{1,2} Domains. Contexts	These sections have been added to provide definitions for following sections. The concept of variables being associated with a particular domain depending on assignment is ambiguous.			

23	816	8_8.3	8.3	Behavioral Interaction	Provides definitions and rules for mixed access and mixed events. Rules for synchronization are not sufficient to produce portable code - needs definition.			
24	817	8_8.3.1	8.3.1	Accessing discrete nets and variables	'Bit' is pretty strange. Results are undefined if any bit values are 'x'/'z'.			
25	818	8_8.3.6	8.3.6	Concurrency	New section(s). [Attempts to] provide rules under which the four subsequent sections are to be interpreted. Totally inadequate to write portable code that accesses variables from multiple domains. This is important since these mechanisms are proposed for constructing connect modules, etc. Whole area requires some clear semantics and possibly additional synchronization constructs.			
26	819	8_8.4	8.4	Discipline Resolution	Totally reworked from previous versions. The algorithms described are based on net types rather than on drivers/loads that appear on the a mixed net. The source of much disagreement. Antrim AMS has a different view based on the importance of the entities on the net rather than the declared net types; and an emphasis on accurate representation of the analog part of mixed nets. I think this whole section will be replaced by whatever a successful AMS simulator decides to implement.			
27	820	8_8.4.3	8.4.3	Connection of continuous-time disciplines	Makes it illegal to have incompatible continuous disciplines on the same net (previously undefined). Antrim AMS allows this; standard rules for connect module insertion apply. User has complete control and it's a useful feature.			
28	821	8_8.5	8.5	8.{5-8} Connect Modules	Modified from previous versions. New syntax (connectmodule). New mechanism for finding matching connect modules. New block added (connectrules/endconnectrules). New mechanism provided for reusable connect modules. New syntax provided for uni-/bi-directional connect modules. 100% incompatible with previous approach. Some of these changes are cosmetic. Others apply totally different mechanisms from prior LRM. Will need major migration plan to accommodate present users.			
29	822	8_8.10	8.10	Driver Access and net resolution	New material or better defined relative to previous versions. Note that these are stated to be callable from connect modules only. These functions allow most of a mixed signal resolution function to be implemented. There are some missing functions, though (e.g. there is no way to discover what change caused driver_update() to become active). net_resolution() is pretty strange - what does 'the default' mean in the explanatory text? Preferred approach is to insert CBs corresponding to net loads/drivers, not to dream up smart CBs.			
30	823	8_8.11	8.11	8.11 Supplementary driver access functions	Added on demand to provide at least a back door to enable accurate registration of analog and digital events (i.e. start analog ramp so that threshold crossing matches digital event). Necessary but not sufficient (mechanisms for decoupling digital drivers from the mixed net, etc, are also required). This whole area needs work - perhaps with user community who care about backannotation, mixed delay calculation, etc.			
31	824	8_9.2	9.2	9.2 Mixed-signal simulation cycle	The information provided about initialization is inadequate to a mixed signal user. This should be written in terms of Verilog-D initialization semantics, and explain any values taken by Verilog-D entities during this phase.			
32	825	8_10	10.0	10 System tasks and functions	Some changes from previous versions. The $realtime vs. $realtime(N) mess gets cleaned up by introducing $abstime(). There is potential ambiguity with system tasks with the same name in Verilog-D and Verilog-AMS, but with two different definitions ($random, possibly); and with system tasks that modify their argument (for example, some system tasks take and modify a seed value. Does this count as 'assignment' for the purpose of defining the domain of the variable (c.f. 8.2.1)? If so, does this behavior apply to user system tasks also? How can the simulator determine whether a system task argument is read-only or is modified by the task?). Verilog-AMS needs table lookup functions for modeling - add $table & variants here.			
33	826	8_11	11.0	Compiler directives	Extended from previous versions. Presumably these are reset by `resetall.			
34	827	8_D	AnnexD	Standard definitions	The file names have changed from <foo>.h to <foo>.vams. When? Why?			
35	828	8_E	AnnexE	SPICE compatibility	Successful implementations will need to be able to import large chunks of legacy SPICE (including models). The material in the LRM is very Spectre-specific (see the independent sources, for example). Additional mechanisms (in particular, global parameter and node features) are necessary to import SPICE code that uses these, since Verilog does not have these capabilities.			
36	829	8_F	AnnexF	Discipline resolution methods	This was added as part of the discussion over net resolution, with the intention of removing details of the algorithm from Chapter 8, and leaving that chapter generic. It didn't work (Annex F looks really odd. One solution would be to simplify Chapter 8, leaving room for alternate views on mixed nets, and delete Annex F).			

37	901	9_12	3.2.1	Coercion of strings to real allowed but not defined	module pureD; parameter real rparam = "string1"; parameter integer iparam = "string2"; parameter sparam = "string3"; initial begin $strobe("String Value assigned to real: \n PCg= %g \n PCf= %f \n PCna= ",rparam,rparam,,rparam); $strobe("String Value assigned to integer: \n PCs= %s \n PCd= %d \n PCf= %f \n PCna= ",iparam,iparam, iparam,,iparam); $strobe("String Value assigned to sparam: \nPCs= %s \n PCd= %d \n PCna= ",sparam,sparam,,sparam); end endmodule The above results in the following: String Value assigned to real: PCg= 1.76884e+09 PCf= 1768843057.000000 PCna= 1768843057 String Value assigned to integer: PCs= ing2 PCd= 1768843058 PCf= 1768843058.000000 PCna= 1768843058 String Value assigned to sparam: PCs= string3 PCd= 32497657065662259 PCna= 32497657065662259 The LRM implies that "string1" should be coerced to a real but does not describe how the coercion should be done. It makes a lot more sense that this would be an error. That would be far more useful for users. Similarly parameter integer iparam = "string2" should be an error. Recommendation: Someone must better define this or we should make illegal. Note, this is certainly specified in 1364 so we			
38	902	9_22	3.2.2	When to do range checks?	Should range checking be done on default values or only final values of an instance. Recommendation: Checking should only be done on the final values as this feature is meant for users to set values not the model developers.			
39	903	9_20	3.4.2	Connections to port expressions (whats a driver?)	Behavioral expression attached to ports - should be able to indicate the disciplines of such ports somehow. e.g; module foo; reg a; reg b; Bar b1(a&b); endmodule How can the discipline of 'a&b' be indicated? Recommendations: Cadence recommendation forth coming, several potential options			

40	904	9_21	3.4.2	OOMR disciplines on behavioral nets	Should be limitations on OOMR declarations of nets that are used behaviorally - should be only able to OOMR to a undeclared net. Therefore couldn't change the discipline of a net that was used behaviorally.
					e.g;
					module top;
					pll pll1();
					mechanical pll.f; // this should be illegal!!
					endmodule
					
					module pll (f);
					electrical f;
					analog begin
					V(f) <+ sin(w*$abstime);
					end
					endmodule
					V() is not the mechanical access function, it is the electrical access function, so is V(f) an error?!!
					
					Recommendation: Make it illegal to use OOMRs to override the discipline of nets that are behavioral. Other nets should be able to be overridden to aid coercion.			

41	905	9_38	3.4.3.2	neutral disciplines	Why do you need neutral disciplines if wire is already neutral - should remove this feature? Recommendations: Consider as part of discipline compatability issues of Annex E3			
42	906	9_8	3.4.3.3 ...	LRM cleanup issue: TRI and WIRE are aliases	Recommendation: Specify that tri should be treated as wire at least in 3.6, other places?			

43	907	9_11	3.5	Initial value of wreal nets not defined	The LRM says.. "If no driver is connected to a wreal net, its value shall be 0.0" It does not define the value of a real net at t = 0. And of course a net cannot store a value (except trireg), its value as we know is only determined by its drivers. If in an example however, we have a driver (continuous assignment.) to the real net. In the same example, if I removed the continuous assignment, at t = 0, out2 = 0. I am not sure as to what the value of out2 should be (at init.) when it has a driver. Recommendation: value set to 0.0 if value hasn't been determined at t=0.			

44	908	9_5	3.5 and 7.7.3	Real value port examples have errors:	The examples in 3.5 and 7.3.3 have errors that prevent them from working without additional changes. The example in 3.5 should be changed as follows:
					// The following three lines should be added so that a wreal is passed into foo
					wreal wstim;
					assign wstim=stim;
					foo f1(wstim,load);
					// foo f1(stim,load); //This line should be deleted3 as it is illegal for ports of type real
					// dut d1 (load, out); // This line should be deleted as it provides not added value
					The example in 7.3.3 should be changed as follows:
					First there is no top level module that instantiates the two blocks so add:
					module top ();
					wreal stim;
					reg clk;
					wire [1:8] out;
					teststim tb1 (stim, clk);
					a2d dut (out, stim, clk);
					initial clk=0;
					always #1 clk=~clk;
					endmodule
					
					In addition, the testbench module must be converted to use wreal since it is passing a real value through one of its ports (the whole reason for
					wreal). The following fixes this issue:
					module teststim (wout,clk); // change output port to wout
					input clk;
					output wout; // change output port from out to wout
					real out;
					wire clk;
					wreal wout; // add wout declaration as type wreal
					assign wout=out; // assign wreal (wout) value to be value of real (out)
					....
					Recommendations: Make the above changes as shown			

45	909	9_32	3.6	default_discipline clarifications	In the first paragraph, it seems that the word scope in used in two different ways which is potentially confusing; 1. as the scope of application of the compiler directive 2. as an argument to the default_discipline compiler directive. Also does the scope argument only apply to the refer instance or does it apply to the children of that instance too? Recomendation: Clarify this paragraph to make clear			
46	910	9_33	3.6	default_discipline only for digital?	Is default_discipline only applicable to digital? If so then need to remove references to 'default_discipline electrical e.g. p3-20 of 2.0 LRM. Recomendation: Resolve analog default_discipline (section 11.1) and then ensure that this section is in alignment.			
47	911	9_31	3.7	Discipline presendence issues	'default_discipline as a compiler directive seems like a very poor approach for library-based simulation and for simulation based on configurations. It makes much more sense to put this information into a design unit such as connectrules or the config. It is also more consistent with the way configs, connectrules are handled. The compiler directive 'default discipline is not a very suitable way to specify how a hierarchy flattening action (discipline resolution) is to be done. The compiler directive 'default discipline is not a very suitable way to specify how a hierarchy flattening action (discipline resolution) is to be done. Okay maybe someone could use the scope, qualifier fields of the default_discipline. However these refer to instance names and there is no precedence in compiler directives for doing that. Instances haven't been created at compile time. How are conflicting default_discipline references to be resolved? e.g. in one part of a file there is; 'default_discipline electrical top.foo.bar and in another part of the file there is; 'default_discipline mechanical top.foo.bar Recommendation: Items 3 and 4 in the precendence list (those referring to instances) should be removed from the list. For			
48	912	9_41	3.7	disiplines rules of branches	What are the rules for decid-ing the discipline of Branches? - LRM isnot clear on this. Section 3.7 Recommendation: Someone needs to define this.			

49	913	9_37	3.9	branches - clarifications	- should say that branches cannot be declared using discrete nets. - Clarification of vector branches is needs 1) It should be illegal to create a vector branch from Vector terminals of different sizes. 2) It should be illegal to create a vector branch from Vector terminals of different directions or else it should be specified how they are connected up. 3) When a vector branch is created from vector nets, it range size (direction?) should be the same as the vector terminals. Recommendation: This feature either be clarified or removed			
50	914	9_42	4.5.1 & 6.7.4	Initial Conditions	What mechanism should be used to set initial conditions; (analysis("ic")) or @(initial_step("ic"))? If so how does it work and if so, how will a piece of code like the following behave; @(initial_step("ic")) V(out) <+ V(in); Recommend that this feature be removed or clarified			

51	915	9_24	5.1.6	Implicit Switch Branches?	What is the behavior of
					if (open)
					I(p,n) <+ 5;
					Is this equivalent to;
					if (open)
					I(p,n) <+ 5;
					else
					I(p,n) <+ 0;
					Recommendation: These should be considered the same, clarify in LRM.			

52	916	9_25	5.3.2	Indirect assignments in conditionals	Indirect branch assignments should be illegal inside conditionally executed statements. The LRM doesn't state this and doesn't indicate what behavior should occur if it happens; e.g.; analog begin if (xx == 2) then V(out) :ddt(V(x)) == 0; end Recommendation: This should be stated as illegal, like other conditionals			
53	917	9_27	6.1 & BMF	Syntax consistencies with 1364	To consistent with 1364 formulation, there should be semi-colons after: analog_branch_contribution analog_indirect_branch_assignment analog_procedural_assignment procedural_assignment Recommendation: Make above changes			
54	918	9_28	6.3, 6.4, 6.5, BNF	(28) Syntax 6-3, Syntax 6-4, Syntax 6-5 and BNF	These should contain no semi-colons after changes to syntax 6-1 above. Recommendation: Make above changes			
55	919	9_26	6.4	Switch branches illegal in BNF	The BNF of the conditional_statement disallows switch branches. In 6.1, strongly recommend that attempts to limit analog_statements inside analog statements using BNF be removed and replace by a semantic restriction. It is impossible as far as I can tell! See restrictions on analog operators in 4.4.1 as this also implies that switch branches are illegal. Recommendation: BNF should be adjusted to allow these. May need to do limitations on conditionals as semantic rules.			
56	920	9_19	7.2 and 1364	defparam vs. instantiation precedence	This comes up in netlisting as the 1364 LRM is really weird in this space. The defparam precedence is defined by last one seen (like a compiler directive) which in a netlisting environment sucks. If searching libraries then the result stated by 1364 is unknown, yek! This should be by level in the hierarchy and then instantiations should be treated as the same as defparams. Recommendation: If this does not get cleaned up to be reasonable we will need to ensure that the issue is addressed when global design variables are supported.			
57	921	9_23	8.2.4	Compatible disciplines	Compatibility of continuous disciplines on the same signal. Should be stated that the continuous disciplines of a signal must all be compatible as they are solved as the same node. Recommendation: Make the above changes to the LRM			

58	922	9_14	8.3.6.4 and 8.3.1	Clarification on X and Z .	These sections should be enhanced to ensure that users and implementers understand that when accessing a digital net, X and Z must be dealt with prior to assigning a value to a variable and certain functions like case, casex, casez, ===, and !== need to be used to prevent errors. We need to close on if == and != can be used in analog on digital signals and if so does X or Z mean a failure of the comparison? Supporting them would require being able to do comparisons against X and Z only. The bottom line is analog cannot be assigned a value of X or Z. Do we need to allow the user to specify what X and Z would be set to? Recommendation: Currently we should limit analog to not support == and != when the signal is digital and contains X or Z. We should clarify this more in the LRM with the core agreement that signals cannot be set to X or Z in analog.			
59	923	9_1	8.4.4.1 and Annex F	Discipline Resolution: No clear definition on how to deal with "leaf level" wires.	Leaf level wires (net segments) are primarily the result of alias modules used in netlisters and pass through's as a result of synthesis. These wires (net segments) have no components connected to them at the specified hierarchy thus the only connections to this wire (net segment) are higher in the hierarchy. While in the detail resolution the discipline is and can be passed down into this net segment if needed the same is not true for the default method. In default all disciplines are passed up the hierarchy but these net segments may not have a discipline to pass up. Recommendation: The default method needs to have additional clarification that this special cases must resolve their discipline by looking up the hierarchy until a discipline can be defined. If two or more ports are connected to this leaf level wire that would pass down a different discipline the basic rules apply, disciplines must be compatible and continuous wins over discrete. Cadence will provide the update for section 8.4 and the Annex			
60	924	9_2	8.4 and Annex F	Discipline Resolution: No clear definition on how to deal with out of module references (OOMRs).	OOMR connections such as (.out(top.middle.bottom.in)) must be considered in discipline resolution. There are two options in addressing this issue. a) OOMR reference recieves discipline from connection b) OOMR connection recieves discipline from referenced net In one the referenced wire (top.middle.bottom.in) is impacted by the connected wire (out) discipline. In the second case the wire (out) is impacted by the resolved discipline of (top.middle.bottom.in) Recommendation: The connection should drive the discipline of the OOMR referene. Cadence will provide the update for section 8.4 and the Annex			
61	925	9_29	8.6	bi-dir issues	The text suggests that a bidir can be connected to a port that is being driven by a reg or an expression.This doesn't make an sense! Recommendation: This needs to be fixed as a reg can drive but cannot be written to from outside the module.			
62	926	9_7	8.10.5	net_resolution function: No one liked this so if we are going to change it lets do it now.	We have used the assign dVal = dVal; without much problems or complaints. The real issue is explaining driver receiver segregation not the syntax. Also, in 8.10.5 if we keep this syntax we should change it to be net_resoultion(port_identifier, net_identifier). LRM needs to clear up some things here i.e. 1) assign d=d will not work if d is a reg, 2) net_resolution is pretty hokey. Recommendation: Get rid of net_resolution and move back to assign statement. With connect modules now marked by the connectmodule keyword we should consider making assign dVal=dVal a default (not required to be specified) and then do a better job documenting how all of this works. What about if dVal is a reg?, does it get segregated in a connect module? Probably not. Should reg's be allowed on connect module ports since they cannot "listen"? suspect this is needed so need to resolve this			

63	927	9_36	8.11	Supplementary drivers and delays	Supplementary driver_update functions are insufficiently well defined in terms of what delays should be accounted for?
					a = #1 b;
					#1 a =b;
					must both of these be accounted for? How about SDF delays?
					How about;
					#1;
					$strobe("testing");
					a = b;
					Recommendation: These should be removed, made informative or more clearly defined.			

64	928	9_42a	9.0	Which solver starts first?	Initialization sequence of the simulation should be described - does the digital kernel or the analog kernal go first? Recommendation: Specify that digital start first as part of the VHDL-AMS sync effort. Needed for simulator commands at time zero as well as mixed language simulators			
65	929	9_42b	9	Initialization method - Different from VHDL	With the movement to mixed language simulators this is an issue and while we could say it is therer problem I think we need to address it. We can either switch to theirs or we can specify both methods and let the user select between them. For circuits with VHDL we might be forced to use only theirs. Recommendation: Specify the same method as VHDL-AMS. Needed for simulator commands at time zero as well as mixed langauge simualtors See Figure from VHDL Manual or Web Site			
66	930	9_15	10.2	1364 sync-up Random function	Random function for analog not clearly defined but in 1364 it is as the code is now provided. Recommendation: Sync up with 1364-2001 ASAP			

67	931	9_39	12.5.2	VPI Issue	Nature and discipline should not use param_assign, instead there should be a new object created called attr_assign. Recommendation: Make the above change			
68	932	9_4a	Annex A	BNF clarification - Connectrules	LRM states that the connect statements can have 'zero or more' count in the connectrules block. For example: connectrules AMSconnect; endconnectrules Recommendation: Either allow (might be usefule for tools that create these rules) or changed to one or more.			
69	932	9_4b	Annex A, 8.5	BNF clarification - Connectmodule	8.5 is missing net_resolution and both are missing digital sections at a minimum. Someone needs to look at this. Recommendation: Add the missing data.			
70	933	9_18	Annex B	flow and potential, should these be global keywords?	May not be a big of an issue after 1364 pulled the rug out from us on the section specific keywords. Should we go back to one single list? Recommendation: Consider impact of single list of keywords. Must move potential and flow to global keywords (from B2 to B1) as they are needed in accessing attributes (see 5.2.2)			
71	934	9_3a	Annex C	Annex C changes that were missed: Null Argument	In VerilogA 1.0 the following was allowed: {} while in VerilogAMS 2.0 null arguments are defined by ", ," (comma-nullarg-comma). Thus {} would be represented as {,}. Null args for things like laplace and z-transforms of no numerator must be written as either { 1 } or { , }. Recommendation: This needs to be added to the list of changes and possibly shown in the areas where most likely to occur such as in the laplace and or z transform sections.			
72	934	9_3b	Annex C	Annex C changes that were missed: Z-filter roots	The original 1.0 LRM version had the roots specified as the product of terms like: (1) (1 - z^(-1)/(r_r + j r_i)) ...so the poles and zeros are roots of the polynomial in z(-1), which makes sense given the zi_nd form is a polynomial in z(-1) -- to get your poles and zeros you just factor the polynomial. But, the 1.4 LRM has: (2) (1 - z^(-1)*(r_r + j r_i)) ...which means that the term goes to zero if z = (r_r + j r_i)), so the roots are of a polynomial in z, the inverse of the above. Recommendation: Researching to determine why this change was made. Need to find out reason for change to determine if going back is even feasible. Need to add this to table also.			
73	934	9_3c	Annex C	Annex C changes that were missed: boundstep argument	Verilog-A defined the argument to $bound_step() to be constant. Verilog-AMS allows "expression" as an argument to $bound_step() which can be dynamic. Recommendation: Add this Verilog-A 1.0 typo to the list of changes in Annex C.			
74	934	9_3d	Annex C	Annex C changes that were missed: $random interpretation	One may argue that only our implementation will change as others interpreted the Verilog-A LRM to be what is in 2.0 but in either case it was not clear. Also, we will be changing to match up with 1364 meaning shortly. Recommendation:Add note in Annex C.			
75	935	9_16a	Annex D	Discipline and Constants file corrections:	The units "coul" should be "C" for nature Charge. This is the standard SI symbol. Of course, that could confuse some folks with temperature would be "degC". Recommendation: Make the above change			
76	935	9_16b	Annex D	Discipline and Constants file corrections:	The unit on Angle would be "rad" to be SI compliant. And "rad/s" for Angular_Velocity and "rad/s^2" for Angular_Acceleration. Each of these put an "s" at the end of rad. Recommendation: Make the above change			
77	935	9_16c	Annex D	Discipline and Constants file corrections:	The physical constants should be listed with a reference. What is the source of these values? The 1998 NIST values differ from those given for physical constants: charge: 1.602176462e-19 light: 2.99792458e-8 boltzmann: 1.3806503e-23 planck: 6.62606876e-34 ...etc... Source:physics.nist.gov/cuu/Constants/index.html Recommendation: Make the above changes			
78	935	9_16d	Annex D	Discipline and Constants file corrections:	Both Boltzmann and Planck are misspelled in the constants file. Recommendation: Make the above changes			
79	936	9_10	Annex D:	upcase issues with disciplines.vams file	There is also a clash between the nature "Force" and the Verilog keyword "force" when doing -UPCASE. In addition each nature like Voltage is case sensitive (note uppercase "V") which when used in -upcase cannot work unless we define something else. Recommendation: Leave as is but provide a warning to the users about these conflicts.			
80	937	9_34	Annex E	SPICE master name conflict	If a SPICE master has the same name as a Verilog master, which matches or is it an error? Recommendation: Thoughts? Do we allow overriding of analog primitives? Will the LRM force a specific implementation?			

81	938	9_35	Annex E2	Case sensitive SPICE simulators	Some spice simulators are case-sensitive, this should be accounted for too. Issues exists with the case-insensitive matching of SPICE components. Verilog is a case-sensitive language and if I write a Verilog construct (in this case an instantiation), it should comply with with the rules of Verilog i.e. case insensitivity. Believe it is better and more consistent with Verilog if all SPICE references must be lower-case, then the binding algorithm is much less complex. Consider that if I define a model called `Cap' and a module called `cap'. However if I type `cap' and expect to get `Cap', I would be wrong. Recommendation: Require all SPICE references to be lower case.			
82	939	9_9	Annex E3	Disciplines of analog primitives: How to set and defaults (see 9_30)	Analog primitives cannot via the language get a discipline defined so we need to specify a default value and a method for setting the disciplines. Recommendation: Lots of possibilities, part of discipline resolution method, use OOMR declarations, default_analog_discipline (of course compiler directives suck in most of today's solutions) See more on next item			
83	939	9_30	Annex E3, 3.8	Compatible disciplines to analog primitives (see 9_9)	Certain primitives are not limited to electrical domain. Primitives like sources, resistors, capacitors, ... can often be used in mechanical and other domains but if they are electrical then they will not be compatible. Recommendation: Analog primitives should default to domain continuous (neutral) and the discipline should resolve as follows: (THIS NEEDS MORE WORK!) i. The discipline of defined primitives and behavioral blocks connected to each port - If incompatible disciplines exists then error ii. The global analog discipline iii. The default discipline The above assumes that the default discipline will be electrical for continuous domains and that there is a mechanism(s) for setting the global discipline and the discipline of specific ports. Disciplines of ports could be set via OOMR discipline declarations (mechanical top.I1.I2.R1.a) where a is the port name.			

84a	949	9_13a	see items	LRM Cleanup Typos	a. Section: 3.4.1.1 and 3.4.3.5 : A user define attribute is specified called "max", this is a keyword and not be used here. Should change to something like "maxvalue".
					
					b. Section: 4.4.14, table should have absdelay not delay
					
					c. Section: 6.7.4: In table 6-1, @final_step for DCOP should be 1 not 0.
					
					d. Section: 8 : Several places have the keyword merged listed as merge
					
					e. Section 8.2.3 : The figure has Net C.b_out which should be Net C.c_out
					
					f. Section: 8.3.1, 8.8 : Mixed signal examples falsely use == comparison of digital signals in analog context. Need to change examples to use methods that support X and Z.
					
					8.3.1 Example:
					.....
					real aout;
					analog begin
					if (in === 1)
					aout = 3.0;
					else
					aout = 0.0;
					V(out) <+ aout;
					end
					endmodule
					
					8.8 Example:
					.....
					analog
					V(e1) <+ transition((cm === 1) ? 5.0 : 0.0);
					.....
					
					Recommendations: Make the above changes			

84b	940	9_13b	see items	LRM Cleanup Typos II	g. Section: 8.3.2 : Next to last paragraph is in error, change to: .... and statements that are .... casez, shall report ... Add the that and drop the which. h. Section: 8.6: p8-17, Last line should be change to "and whose other connection is compatible with electrical" i. Section: 8.10.6 : In the example on 8-37 of the 2.0 LRM, has input and inout as the directions for the ports of the CM. They both had to be inout if one is. j. Section: 8.10.6 : The example has "initial net_resolution(d,out);" initial is not needed k. Annex C : The Table C-1 first item should be $abstime not $atime. Recommendations: Make the above changes			
85	941	9_40	Global	Support for global design variables (accessible throughout hierarchy)	Can do some of this via defparams and OOMR but can get very ugly and many limitations that make this difficult to be considered a viable solution as-is. Recommendation: Need to consider the dynamic parameter proposal. Lots of issues with current capabilities that this must resolve.			
86	10	10	3.8	Issues on discipline and Nature compatibility	Section 3.8 of LRM 1.9 states 1. The opening paragraph begins by discussion of net compatibility. However, it then states to conclude: "The following rules apply in deciding whether two disciplines are compatible". Some of the rules presented are not about disciplines per se, but rather about net usage, which then contradicts what the statement has. 2. There is no rule to state that a nature is compatible with itself. 3. There is no rule to state that a nature is compatible with its base nature. 4. The Nature Incompatibility rule does not make sense. A. If the natures are compatible, of course they will be not incompatible! B. A nature cannot be compatible with a non-existent nature. For a nature to be non-existent, it must not be defined. The correct wording of this rule should in fact refer to a non-existent binding within the discipline. 5. The Potential and Flow Compatibility Rules state effectively the same thing. It would be better simply for each to state that if the potential or flow natures are incompatible, then the disciplines themselves must be incompatible. This is much clearer and more concise. 6. Empty Discipline Rule is misleading: an empty discipline has no domain, but rather is compatible with any other discipline, regardless of domain. 7. Discrete Domain Rule is actually applying to nets; disciplines (discrete or otherwise) do not have a signal value type. 8. Signal Connection Rule is also applying to net usage. This is also two different rules which are tenuously linked, but have been combined. The first rule states 9. The rules themselves are scattered and intermixed. It would be much better to separate them into groups of rules about net connections, groups of rules abo Proposal: 3.8 Net Compatibility Certain operations can be done on nets only if the two (or more) nets are compatible. For example, if an access function has DISCRETE DOMAIN RULE: Digital nets with the same signal value type (ie. bit, real, integer) are compatible with each other if their disciplines are compatible, i SIGNAL DOMAIN RULE: It shall be an error to connect two ports or nets of different domains unless there is a connect statement (see 8.4) defined between th SIGNAL CONNECTION RULE: It shall be an error to connect two ports or nets of the same domain with incompatible disciplines. The following rules shall apply to determine discipline compatibility: SELF-RULE (DISCIPLINE): A discipline is compatible with itself. EMPTY DISCIPLINE RULE: An empty discipline is compatible with all other disciplines, regardless of domain. DOMAIN INCOMPATIBILITY RULE: Disciplines with different domain attributes are incompatible. POTENTIAL INCOMPATIBILITY RULE: Disciplines with incompatible potential natures are incompatible. FLOW INCOMPATIBILITY RULE: Disciplines with incompatible flow natures are incompatible. The following rules shall apply to determine nature compatibility: SELF-RULE (NATURE): A nature is compatible with itself NON-EXISTENT BINDING RULE: A nature is compatible with a non-existent discipline binding. BASE NATURE RULE: A derived nature is compatible with its base nature. DERIVED NATURE RULE: Two natures are compatible if they are derived from the same base nature. NATURE COMPATIBILITY RULE: Two natures are compatible if they have the same value for the access and units attribute.			

87	11	11	6.4	Issues on if-elseif	Consider the following Verilog-AMS module:
					
					module curly;
					genvar g;
					integer i;
					electrical a,b;
					analog begin
					if(g==3) // #1
					V(a) <+ transition(...); // #2
					else if(i==6) // #3
					V(b) <+ slew(...); // #4
					else // #5
					// some other code
					end
					endmodule
					
					Now, the LRM defines any if() with a genvar expression as an analog if. g==3 is a genvar expression, so conceivably the if from #1 with else at #3 is an analog if statement; the if at #3 with else at #5 would then be considered as a nested if statement. In this scenario, statement #2 is OK but #4 is wrong. This is fine if you consider if-else as an integral statement, which the LRM kind of says.
					
					However, the LRM also defines if-else-if-else as a "multi-way decision". Under this situation, the whole statement from #1 to #5 is a *single* statement. Because the condition at #3 is not a genvar expression, the whole thing is procedural, not analog, and therefore both #2 and #4 are wrong.
					
					We have a case here of the LRM contradicting itself, and the consequences are pretty bad for analog operator usage			

88	12	12	3.8	Issues with regards to example on Section 3.8	The examples for discipline compatibility need to be re-worked; there are quite a few issues: 1. discipline highvolt is shown to be derived from discipline electrical. This is wrong. highvolt's declaration should be: discipline highvolt potential Voltage; flow Current; potential.abstol = 1; enddiscipline 2. The fourth dot-point giving a description should be changed. Discipline mechanical (which does not appear in the example) should be changed to rotational (which is in the example. 3. The sixth dot-point makes no sense at all in the context of explaining the example. It should state the following: * discipline empty is independently compatible with all other declared disciplines, ie. electrical, highvolt, sig_flow_v, sig_flow_i, rotational, sig_flow_x, sig_flow_f and logic. This is because empty has no natures and has no declared domain. 4. The final dot-point also needs some adjustment. The point begins discussing the disciplines and their compatibility, but then states that "A connect statement must be used to connect these nets and or ports together". This final statement should state "A connect statement must be used to connect nets or ports of these disciplines together." 5. There is no discipline with an explicit continuous domain specification. There should be one, perhaps as follows: discipline continuous_elec domain continuous; potential Voltage; flow Current; enddiscipline The explanation should have an extra dot-point similar to this: * Disciplines electrical and continuous_elec are compatible because the default domain for electrical is continuous, and the s			
89	13	13	ENH	Add support for NaN & 'X'	I would like to introduce NaN into Verilog-AMS for a couple of reasons: For initialization of 'real' type values. Any real arithmetic function whose result depends on an 'X' (or 'Z') digital value should have the result NaN. It will be illegal to assign NaN to a branch, but the user can test for NaN e.g.: if (Vout != NaN) V(out) <+ Vout; This may help later with VPI functions that return can return NaN.			

90	14	14	3.8 ENH	Discipline Compatibility	Discipline compatibility for disciplines derived from the same base discipline depends on their attribute compatibility. If attributes differ (e.g. abstol) then thereshould be a resolution function that takes all the values present and returns the working value, otherwise the disciplines should be considered incompatible.
					
					Attribute resolution functions probably need new syntax e.g.:
					
					discipline electrical resolve abstol = abs_min; // probably in the standard include
					or
					discipline electrical resolve my_attr = my_func; // my_func is bound later
					or
					discipline electrical resolve my_attr function begin
					//implicit integer my_attr_size
					integer n = my_attr_size;
					real tot = 0;
					while (n-- > 0) tot += my_attr[n];
					resolve = avg/my_attr_size; // return average
					endfunction
					
					If you have two electrical disciplines with different vdd/vss attributes and without a resolution function for them they would be considered incompatible, and the Verilog-AMS compiler will look for a discipline connection rule (which might be a level shifter module).			

91	15	15	11.5	`include	Proposal Unlike Verilog-D, Verilog-A has "system" include files for default/standard disciplines physical constant values and maybe simulator/tool supported features. As with "C" Verilog-A[MS] should include these with "<>" rather than "" quotes. For backward compatibility the system file include path should be appended to the end of the user search path so that system files quoted with "" will be found (if not overridden).			

92	16	16	9.0 ENH	Mixed signal initialization (digital)	Existing Verilog-D simulators are entirely transient in operation, and therefore don't have any mechanisms for static/steady state simulation. Clearing all the time-zero events won't necessarily help stabilize feedback loops through analog behavior.
					
					For gate-level combinational logic it is possible to evaluate valid steady-state values and therefore close loops, but synchronous logic and behavior usually requires a clock cycle or reset pulse to bring it to a sensible state.
					
					Proposal
					
					We should add a block type to Verilog-D for evaluating module steady-state behavior. We need a syntax which allows multiple event driven processes e.g.:
					
					module stt_mc(clk,a,b,c,d,q1,q2);
					....
					steady begin
					@(a,b,c,d) begin
					{q1,q2} = chk_func(q1,q2,a,b,c,d); // only allow valid states
					end;
					endsteady;
					
					"steady" blocks would use any Verilog statements that don't involve delays (or delays would be ignored). Data values from the steady-state analysiscould/would be discarded after operating point analysis and 'X' values used instead.
					
					N.B. This kind of functionality may be required for arbitrary operating point analysis.			

93	17	17	ENH	Filters for foreign languages	Most CAD design systems use legacy design languages (e.g. Spice - which has many variants, Spectre and Mast). Stipulating that Verilog-AMS simulators should read any of these languages directly is an unreasonble onus on simulator developers, even if the DOD requires Spice compatibility. It is also unreasonable (and sometimes impracticable) to have users maintain multiple copies of the same data in different formats.
					
					Proposal
					The "external module" proposal should be extended with parameters and/or keywords indicating a source file, the language of the source file and a filter program which will translate the specified source into Verilog-AMS. Filter programs could be supplied by users or by vendors.
					
					Example:
					extern module my_spf;
					parameter source = "/proj/big_chip/big.spf"; // may be list
					parameter language = "HSpice";
					parameter filter = "spc2vams"; // may include arguments
					endmodule
					
					If the simulator can read the specified file type directly (and the file exists) it may do so, otherwise a "pipe" is created and the filter called (see Unix "popen") as:
					<filter> -module=<module name> -language=<language>\
					-simulator=<caller name> -version=<caller version>\
					<source>
					e.g.:
					spc2vams -module=my_spf -language=HSpice\
					-simulator=vams -version=0.99.1\
					/proj/big_chip/big.spf
					
					The output of the filter should be read in the same manner as a `include'd file.
					Since the filter program may need extra information, it's standard input stream should be the (post processed) text from exte
					
					If the filter returns a non-zero status the simulator should abort.
					Notes: The source specification does not have to be a file, it can be a database reference (e.g. milkyway) or some other log			

94	18	18	ENH	Light Weight Conversion	Rationale Current "connect module" insertion only addresses automatic conversion of signals passed through ports (structural connection). This is viewed as a "heavy weight" conversion problem requiring persistant state (hence the use of modules) and provides high-accuracy multi-domain signal resolution. Behavioral code (e.g. test benches) often ask for values which are not passed through ports (e.g. OOMRs) but may be in another domain in a mixed signal design (unknown to the testbench designer). These conversion requests are often just "probes" that need neither resolution or persistant state and are viewed as "light weight". Proposal As light-weight conversion doesn't require persistant state it can be performed by Verilog functions. An analog to digital conversion function would take the potential or flow of a branch as an input and return a logic value (0,1,X,Z + strength), and a digital to analog conversion function would convert the drivers or resolved value of the digital signal to a real value. Light-weight conversion would be short-circuited if a signal is converted by a heavy-weight conversion - i.e. the output of the heavy-weight conversion is used instead of using the light-weight conversion function. The syntax for specifying a light-weight "connect" would be similar too the heavy-weight: connectfunction [real] <module_identifier> (<input_declaration>) endfunction The function overloads the connect module name in the connect rules - i.e. if the rules indicated the connect module to be in If a connect rule has only matching connect-functions and no connect-modules then only light-weight connections are possib			

95	19	19	ENH	Representation Stops	In order to avoid having to translate netlists into slightly different forms for different purposes, or doing nasty things with `define/`ifdef, it would be useful to have a "representation stop" in the language (first mentioned by Kim Hailey [Metasoftware] ~1996). A "representation stop" mechanism indicates what a module call represents to different simulator kernels or secondary tools. In particular it allows netlists translated from Spice to be used with different simulator models for transistors or used in a digital simulator with switch level models. An official "representation stop" mechanism also allows different versions to be simultaneously visible (unlike `ifdef) and declares them to be different views of the same object which makes it is easier for a smart tool to check that they are consistent. Proposal A and B to follow			

					Item 95: Proposal A: Augment "external module" and "macromodule" definitions with a "simulator class", a suggested syntax is:
					
					<repstop> ::= (extern module|macromodule)
					[<simulator class::>]<module name> [(<ports>)];
					....
					See also the "external module" proposal.
					
					Usage: Declaring an external "Spice" transistor:
					extern module spice::nmos(drain,gate,source,back);
					parameter model = "NMOS3";
					parameter l = length ; // translate parameter
					parameter w = width ; // translate parameter
					// this component is primitive of the simulator
					endmodule;
					The simulator would use the parameters "model","l" and "w" as stated and calculated from the actual instance.
					
					Alternative "digital" transistor version of the above for pure digital simulation:
					macromodule digital::nmos(drain,gate,source,back);
					rnmos(drain, source, gate); // use digital primitive
					endmodule;
					
					Simulators and tools would ignore class definitions that they do not understand, and would default to a particular behavior as directed by the user (or Verilog 2000 configuration directives, considering the "simulator class" as the view), if they can ha			

					Item 95: Proposal B.
					Seperate represention-stop declaration.
					Syntax:
					<repstop> ::= repstop <module name> [(<ports>)];
					{<parm>}
					{<alias>[[,<alias>]]} endrepstop
					<alias> ::= alias [<tool>::]<alt module name> [(<alt ports>)];
					{<parm>} endalias
					See message #104 for an example.
					
					Usage: The "rep-stops" for a given simulator would be bundled with it as a header file in the same directory as other standard include files, e.g.:
					
					#include <repstop.h>
					Example contents:
					repstop sp_nmos(d,g,s,b);
					alias spectre::mos(d,g,s,b);
					parameter device = "NMOS";
					parameter model = "BSIM3.3";
					endalias;
					endrepstop
					A translated Spice netlist may include a default rep-stop:			