Analysis of Analysis of the "Alternate Proposal on Class Declaration"

Jay Lawrence
Cadence Design Systems, Inc.
3/14/2003

Rather than attempt to respond point-by point to Arturo's analysis I will simply comment on the requirements he extracted and the conclusions he drew to clarify my position for anyone who may be confused. I've left my original in blue and Arturo's comments in red.

_ Requirement 1: Explicit indication that a class object is dynamic.
Agreed

_ Requirement 2: Allow dynamic allocation of structures.
_ Requirement 3: Allow static allocation of classes.

Allow FUTURE dynamic allocation of structures and static allocation of classes. The intent of this proposal was not to add these at this time. The original proposal gave some suggestions on how this might be done but was not intended as a complete specification, this is way beyond what we have time for in SV 3.1.

_ Requirement 4: Enable simple orthogonal extensions of SystemVerilog data types in the future.
The current adding of all new data types as keywords has made this requirement almost impossible anyway so this is stronger than what I am trying to accomplish. I simply want a syntactic difference between dynamically allocated and statically allocated objects. The extensions of string, associative array, etc that exist in 3.1 are static objects of a dynamic size.

(underlining added in following paragraph)

It's clear from the requirements that the proposal does not attempt to solve a deficiency in the language, nor does it establish that a problem exists that couldn't be solved with the existing class paradigm. Moreover, the proposal doesn't add any additional semantic power to the language. As we will show later, everything that can be done with the alternative proposal can be done with the current language, but with a lot less typing and confusion. The proposal's goal is simply an academic exercise that seeks to create "a regular type system" that will enable "simple orthogonal extensions in the future". Whether one agrees with these goals is irrelevant, what really matters is what additional semantic power the proposal provides to end users, and at what cost. It is our contention that the proposal adds nothing in terms of semantic capabilities, and the immediate cost is a much more complex and verbose language.

No attempt was made to solve a deficiency in the semantics of the language, the addition of the capabilities of classes and dynamic memory is clearly needed. However, we do not believe that allowing for future extension of the language is NOT irrelevant. If someone had told me 3 years ago that we would be extending Verilog to this extent I would never have believed it. I am unwilling to gaze into a crystal ball at this time and hold that these other extensions will not be needed.

Conclusion

The addition of classes is a powerful addition to SystemVerilog 3.1. The default dynamic nature of class declarations optimizes the most common use model and leads to a simpler and easier to use language. The type system regularity espoused by the alternate proposal comes at the expense of a much more verbose and complex language.

Only 2 core changes are required, using **ref()** to pass a static object by reference, and declaring a **ref** typedef for your classes and using it instead of the **class** typedef. The first (**ref**()) could event be eliminated if the tools were allowed to do it implicitly as they do it today, but I would not recommend that. The semantics of pass-by-reference are sufficiently different that I think this is advisable. I don't think this constitutes "much more verbose and complex". I would call it explicit and clear.

Finally, as mentioned at the start, the alternate proposal does not attempt to solve a semantic deficiency in the language, nor does it state a problem that cannot be solved with the current class mechanism. It merely attempts to unify several constructs that we feel are best as they are, otherwise ease-of use, performance, and robustness will all be adversely affected.

The proposal as it stands has minimal impact on ease-of-use (change the typedef you use), there is no impact on performance or robustness because the semantics have not been changed, only the syntax.