Hi, Francoise -

Thanks for getting back to us with the Cadence-questions so quickly.

Cliff's take on most of the questions shown below. Awaiting confirmation from Arturo, Doug, Gord and Dave Rich.

We had a discussion internally at Cadence about the proposed changes to program blocks. That discussion generated a few questions which are listed below and which we would like to see answers:

With the new proposed scheme where tasks and functions called from the program block get reactive scheduling semantics whereas if they are called from a design process, they get regular scheduling semantics:

BTW - I believe the above statement answers most of your questions, and answers them more cleanly than what we were dealing with before this proposal was made.

1. NBAs:

1.1 Are NBAs to design variables inside a task being called by the program be scheduled in the Reactive NBA region or the design NBA region

Reactive-region NBA (the key: called by a program)

1.2 What is the behaviour of a NBA to a concatenation of a module variable and program variable on the target of the NBA?

{mv, pv} <= 2'b00;

If called from a module scope, assignment scheduled in the NBA region.
If called from a program scope, assignment scheduled in the reactive-NBA region.
(This would have been messy and ambiguous using old semantics)

2. Called from a foreign language:

What are the scheduling semantics for a SV task called from another language (VHDL, systemC, C, DPI)?

This one is a bit tricky, but I don't think it is very tricky.
DPI-C function called from a module scope - Active region (?? - I don't know if this was well defined before??)
DPI-C function called from a program scope - Reactive region
Even if the C-function in turn calls a task or function from either a module or program scope.

IEEE Std 1800-2005 does not define scheduling for VHDL or SystemC. These must be considered with respect to the next-gen 1800-PAR.

3. Program constructs restrictions

3.1 Since the proposal is that design tasks called from the testbench get reactive scheduling, are such tasks going to be limited to language constructs allowed in programs? For example blocking assignments are not allowed in programs, are we going to have to dynamically check for blocking assignments if the task is called from a program based process?

Calling a module-scope task from a program-scope would require the task content to be legal when called from a program-scope. I can't think of any problems related to this at this time (although there may be

such problems??). You can make blocking assignments from a program but you cannot make a blocking assignment using the clocking block prefix, but this is true in both the module-scope and program-scope, so I don't see a problem yet. Am I missing something?

3.2 Are task functions local variables considered to be program or design variables? Do blocking assignment program restrictions apply to tf local variables?

I semi-defer to PLI experts here. But again, one can make blocking assignments from a program scope (just not clocking-blocking assignments). This may not be a problem?? Is this covered by the PLI regions??

3.3 A task or function can access a static variable defined outside the task or function, are blocking assignment to out of tf static variables allowed when the tf is called from the program?

Sounds like another PLI-expert question(??)

4. Spawned processes

What are the scheduling semantics of fork join processes of a design task or function when the task/function is called from a program?

Again, the answer becomes much more simple with the new paradigm. If called from a program-scope, we are in the Reactive regions for scheduling.

5. Class variables

5.1 Are class variables considered design or program variables?

I believe the restrictions related to design or program variables all go away with the proposed enhancements. Life and the LRM get much simpler.

If assigned in a program, we are in the program scope and we do not consider where the variable was declared. Same is true for the module-scope.

5.2 Are class properties considered design or program variables? If we were going to allow NBAs to class properties, where would these NBAs be scheduled? (There has been some user request to remove the current limitation of no NBAs to class properties so we need to have an answer for the future).

Similar answer. If called from a program-scope, we are in the Reactive regions ("the little loop") and if called from a module-scope, we are in "the big loop" regions. This will answer the question if nonblocking assignments are added to classes.

6. Continuous assignments

Is it legal to have continuous assignments to wires in a program?

- Yes. Program continuous assignments to wires are legal.
- No cb.port continuous assignments are permitted.
- Cb.port assignments require ***clocking drives*** (looks like an nonblocking assignment(<=) but it is a ***clocking drive*** and we cannot use <= with continuous assignments).

7. Program inout ports

7.1 Can a program have inout ports (wires)? There is an example in the LRM of a program with an inout wire port.

- For inout ports, the port must be a wire (not a logic) and then driven by a cb.port ***clocking drive*** from an initial block.
- Continuous assignments and cb.port ***clocking drives*** to the same wire are legal and have normal resolution semantics (multiple drivers).

7.2 How does the program drive the wire?

I had this same question before the clocking-program-scheduling summit a couple of weeks ago.

- A program can drive an inout port with a continuous assignment.
- A clocking block can drive an inout port using a ***clocking drive*** from an initial block (the inout port, of course, has to be a net type).
- It is legal for one or more continuous assignments and one or more cb.port ***clocking drives*** to drive the same wire and they will have normal resolution semantics (multiple drivers).