Proposal: Exported Tasks (revision 4)

Motivation

SystemVerilog 3.1's DPI supports imported and exported inter-language function calls. An imported function is implemented in C, and it is callable from SystemVerilog. An exported function is implemented in SystemVerilog, and it is callable from C.

Users have requested that the DPI export functionality be extended to include exported tasks. This will enable new and powerful modeling techniques.

For example, a testbench environment may be written in C, and make use of high-level function calls to effect hardware transactions in the DUT. Such a test environment could be achieved in SystemVerilog as follows. Start up the testbench by calling an imported function from within a program block. The imported function is the gateway to a complex body of C procedural code. When necessary, the C code calls back into the SystemVerilog DUT by making calls to various exported tasks. Each such task effects a certain hardware transaction, which normally will consume simulation time. The C code is impervious to this time consumption since it is transaction-level code, rather than cycle-accurate timed code.

Another application enabled by exported tasks is the seamless interleaving of C-based models with SystemVerilog models. Simulation tools can now offer modeling environments in which C and SystemVerilog models all live on the same kernel threading system. The models call back-and-forth between each other in order to achieve the user's desired functionality. This is especially useful when moving from a transaction-level C-based verification system into actual RTL implementation. The TLM C testbench can be re-used by replacing key function calls into the DUT with calls to SV exported tasks.

Changes

1) Section 26.1.1
   Rename section to "Tasks and Functions"

   Add a new second paragraph as follows:

   "It is also possible to perform task enables across the language boundary. Foreign code may call SystemVerilog tasks, and native Verilog code may call imported tasks. An *imported task* has the same semantics as a native Verilog task: It never returns a value, and it may block and consume simulation time."

   In the existing third paragraph (which starts "Every imported function needs..."), change all uses of the word "function" to read "task or function". In addition, the last sentence should be rewritten as follows:

   "Imported tasks and functions can have zero or more formal input, inout, or output arguments. Imported tasks always return a void value, and thus can only be used in statement context. Imported functions can return a result or be defined as void functions."

   In the existing fourth paragraph (which starts "DPI is based...", add a concluding sentence as follows:

   "Similar interchangeable semantics exist between native SystemVerilog tasks and imported tasks."

2) Section 26.2.2 erratum
   The final sentence of the section refers to Annex A. This reference should be changed to Annex D.

3) Section 26.3
   Replace all instances of the word "function" with "task or function". Adjust grammar (plurals, correspondences) as needed.

4) Section 26.4, “Imported functions”
Change title to “Imported tasks and functions”.

5) Section 26.4.1, “Required properties...”
Change use of word “functions” to “tasks and functions”.
Change term “function calls” to “task or function calls”

6) Section 26.4.1.1 “Instant completion”
Change title to “Instant completion of imported functions”
Add a new sentence at the end:

“Note that imported tasks may consume time, similar to native SystemVerilog tasks.”

7) Section 26.4.1.3, “Special properties pure and context”
Change all uses of the word “function” to “task or function”.
Similarly, “functions” changes to “tasks or functions”.
However, leave the second paragraph (starting with “A function whose...”) alone, and add the following sentence to the end of the paragraph: “An imported task can never be declared pure.”

8) Add new section 26.4.1.5, “Re-entrancy of imported tasks”
Add:
“Since imported tasks may block (consume time), it is possible for an imported task’s C code to be simultaneously active in multiple execution threads. Standard re-entrancy considerations must be made by the C programmer. Some examples of such considerations include safe use of static variables, and ensuring that only thread-safe C standard library calls (MT safe) are used.”

9) Add new section 26.4.1.6, “C++ exceptions”
Add:
“It is possible to implement DPI imported tasks and functions using C++, as long as C linkage conventions are observed at the language boundary. If C++ is used, exceptions must not propagate out of any imported task or function. Undefined behavior will result if an exception crosses the language boundary from C++ into SystemVerilog.”

10) Section 26.4.3, “Context functions””

Replace the word “function” with the term “task or function”. Similar for “functions”.

11) In Annex A.2.6, and excerpt thereof in Section 26.4.4, “Import Declarations”:
Change BNF as follows:

```
dpi_import_export ::=
        import “DPI” [ dpi_function_import_property ] [ c_identifier= ] dpi_function_proto;
      | import “DPI” [ dpi_task_import_property ] [ c_identifier= ] dpi_task_proto;
dpi_function_import_property ::= context | pure
dpi_task_import_property ::= context
dpi_function_proto ::=
        function named_function_proto (footnotes a, b)
dpi_task_proto ::=
        task named_task_proto (footnote b)
```

Footnote a) dpi_function_proto return types are restricted to small values, as per 26.4.5
Footnote b) formals of dpi_function_proto and dpi_task_proto cannot use pass by reference mode and class types cannot be passed at all; for the complete set of restrictions see 26.4.6.

[Note: These changes are consistent with sv-cc’s LRM-5 adjustment in the 3.1a LRM. (See http://www.eda.org/sv/LRMChanges.html for details)]

12) Section 26.4.4, “Import declarations”
Change word “functions” to “tasks and functions”, except in the last sentence of the 2nd paragraph. Leave that one alone (“Imported functions can return a result or be defined as void functions.”). Add a new sentence after that says “Imported tasks never return a result, and thus are always declared in foreign code as void functions”.

The third paragraph ends with the sentence “An import declaration can also specify an optional function property: context or pure”. Change this sentence as follows: “An import declaration can also specify an optional function property. Imported functions can have properties context or pure; imported tasks can have property context.”

In the examples at the bottom of the section, in “miscellanea”, add a new example like this:

```
import “DPI” task checkResults(input string s, bit [511:0] packet);
```

13) Section 26.4.6, “Types of formal arguments”
Change all uses of the word “function” to “task or function”.

14) Add new section after 26.6 (will be: 26.7), “Exported tasks”

SystemVerilog allows tasks to be called from a foreign language, similar to functions. Such tasks are termed “exported tasks”.

All aspects of exported functions described above in section 26.6 apply to exported tasks. This includes legal declaration scopes as well as usage of the optional c_identifier.

It is never legal to call an exported task from within an imported function. This semantic is identical to native SystemVerilog semantics, in which it is illegal for a function to perform a task enable.

It is legal for an imported task to call an exported task only if the imported task is declared with the `context` property. See section 26.4.3 (Context tasks and functions) for more details.

One difference between exported tasks and exported functions is that SystemVerilog tasks do not have return value types. The return value of an exported task is an int value which indicates if a disable is active or not on the current execution thread.

Similarly, imported tasks return an int value which is used to indicate that the imported task has acknowledged a disable. See section 26.8 for more detail on disables in DPI.

15) Add new section after 26.7 (will be: 26.8), “Disabling DPI tasks and functions”

Add the following paragraphs:

It is possible for a disable statement to disable a block that is currently executing a mixed language call chain. When a DPI import task or function is disabled, the C code is required to follow a simple disable protocol. The protocol gives the C code the opportunity to perform any necessary resource cleanup, such as closing open file handles, closing open vpi handles, or freeing heap memory.

An imported task or function is said to be in the disabled state when a disable statement somewhere in the design targets either it or a parent for disabling. Note that the only way for an imported task or function to enter the disabled state is immediately after the return of a call to an exported task or function. An important aspect of the protocol is that disabled import tasks and functions must programatically acknowledge that they have been disabled. A task or function can determine that it is in the disabled state by calling API function svIsDisabledState().

The protocol is composed of the following items:

1. When an exported task returns due to a disable, it returns a value of 1. Otherwise it must return 0.
2. When an imported task returns due to a disable, it must return a value of 1. Otherwise it must return 0.
3. Before an imported function returns due to a disable, it must call API function svAckDisabledState().
4. Once an imported task or function enters the disabled state, it is illegal to make any further calls to exported tasks or functions.

Items 2, 3, and 4 are mandatory behavior for imported DPI tasks and functions. It is the responsibility of the DPI programmer to correctly implement the behavior.

Item 1 is guaranteed by SystemVerilog simulators. In addition, simulators must implement checks to ensure that items 2, 3, and 4 are correctly followed by imported tasks and functions. If any protocol item is not correctly followed, a fatal simulation error is issued.

Note that if an exported task itself is the target of a disable, its parent imported task is not considered to be in the disabled state when the exported task returns. In such cases the exported task will return value 0, and calls to svIsDisabledState() will return 0 as well.

When a DPI imported task or function returns due to a disable, the values of its output and inout parameters are undefined. Similarly, function return values are undefined when an imported function returns due to a disable. C programmers may return values from disabled functions, and C programmers may write values into the locations of output and inout parameters of imported tasks or functions. However, SystemVerilog simulators are not obligated to propagate any such values to the calling SystemVerilog code if a disable is in effect."

16) Annex A.2.6, Section 26.6
Add new text to Annex A.2.6, and include the excerpt here as is done in section 26.6>
dpi_import_export ::=
| export "DPI" [c_identifier =] task task_identifier;

17) Section D.1, "Overview"
There are two bullet items starting at the 2nd paragraph.
Add two new bullets after the existing two as follows:
— Tasks implemented in SystemVerilog and specified in export declarations can be called from C; such functions are referred to as *exported tasks*.
— Functions implemented in C which can be called from SystemVerilog, and can in turn call exported tasks, are referred to as *imported tasks*.

18) Section D.3, "Portability"
Change "functions" to "tasks or functions"

19) Section D.5, "Semantic Constraints", and Section D.5.1
Change "imported functions" to "imported tasks and functions".
Change "exported functions" to "exported tasks and functions"
Change "SystemVerilog function" to "SystemVerilog task or function" (4th paragraph)

20) Section D.5.5, "context and non-context functions"
Change "functions" to "tasks and functions"

21) Section D.7.2, "Calling SystemVerilog functions from C"
Change "functions" to "tasks and functions"

Add the following new paragraph at the end of the section:
"Calling a SystemVerilog task from C is the same as calling a SystemVerilog function from C with the exception that the return type of an exported task is an int value which has a special meaning related to disable statements. Please see section 26.8 for details on disable processing by DPI imported tasks and functions."

22) Section D.8, and all child D.8.x sections.
Rename to “Context tasks and functions”
Change “functions” to “tasks and functions” throughout the section (exception: VPI/PLI functions should not be changed)

23) In Annex E.1, add the following two API function descriptions at the very end:

```
/*
 * Returns 1 if the current execution thread is in the disabled state.
 * Disable protocol must be adhered to if in the disabled state.
 */
int svIsDisabledState();

/*
 * Imported functions call this API function during disable processing to acknowledge
 * that they are correctly participating in the DPI disable protocol. This function must be
 * called before returning from an imported function that is in the disabled state.
 */
void svAckDisabledState();
```

