0.1 Inclusion of Foreign Language Code

This section describes common guidelines for the inclusion of Foreign Language Code as well as related PLI and VPI functions into a SystemVerilog application. This intention of these guidelines is to

- enable the redistribution of C binaries in shared object form
- ensure the compatibility for all linkage aspects that would appear in the C code

It is assumed that the provision of a common inclusion method makes it easier to switch between tools, requires less learning and tool training on the user's side and does as such enhance the competition between tool vendors.

Foreign Language Code - as it is referred by this section - is functionality that is included into SystemVerilog using the DirectC Interface. As a result, all statements of this section do apply only to code included using this interface; code included by using other interfaces (e.g. PLI, VPI) is outside the scope of this section. Due to the nature of the DirectC Interface, most Foreign Language Code will usually be created from C or C++ source code, although nothing precludes the creation of appropriate object code from other languages. This section adheres to this rule, it's content is independent from the actual language used.

In general, Foreign Language Code may be provided in form of object code (compiled for the actual platform) or in form of source code. The capability to include Foreign Language Code in object code form is a mandatory feature that must be supported by all simulators according to the guidelines in this section. The capability to include Foreign Language Code in source code form is optional, but must follow the guidelines in this section, when it is supported by a simulator. The inclusion of object and source code is assumed to be orthogonal and must no be dependent on each other. Any interferences between both inclusion capabilities should be avoided.

This section defines

- how to specify the location of the corresponding files within the file system
- how to specify the files to be loaded (in case of object code) or
- how to specify the files to be processed (in case of source code)
- in which form object code has to be provided (as a shared library or archive)
- how to specify compilation information required for processing source code
- how to register provided PLI and VPI functions with a SystemVerilog applications

It does not define

- how to organize the corresponding files within the system (a certain directory structure)
- how object code files are loaded
- how source code files are processed and finally included in a simulation (besides the requirement that this must be fully transparent to the user)

Although this section defines guidelines for a common inclusion methodology, it requires multiple (usually two) implementations of the corresponding facilities. This takes into account that there might be multiple users having different viewpoints and different requirements on the inclusion of Foreign Language Code, e.g.:

- A vendor that wants to provide his IP in form of Foreign Language Code. In this case it is often required that there is a self-contained method for the integration, that still permits an integration by a third party. This use case is often covered by a bootstrap file approach.

- A project team that specifies a common, standard set of Foreign Language code; this set might change, dependent on technology, selected cells, backannotation data and other items. This use case is often covered by a set of tool switches, although it might also use the bootstrap file approach.
- An user that wants to switch between selections or provides additional code. This use case is covered by providing a set of tool switches to define the corresponding information, although it might also use the bootstrap file approach.

This section proposes a set of switch names to be used for a particular functionality. This is of informative nature; the actual naming of switches is not part of this standard. It might further not be possible to use certain character configurations in all operating systems or shells. Therefore any switch name defined within this document is a recommendation how to name a switch, but not a requirement of the language.

Location Independence

All pathnames specified within this section are intended to be location independent, which is accomplished by the switch “-sv_root“. It can receive a single directory pathname as value, which is then prepended to any relative pathname that has been specified. In absence of this switch, or when processing relative filenames before any “-sv_root“ specification, the current working directory of the user must be used as default value.