0.0.1 Inclusion of Object Code

The inclusion of compiled object code is required for cases where the compilation and linking of source code is fully handled by the user; thus only loading of the created object code is needed to integrate the foreign language code into a SystemVerilog application.

It must be supported by all SystemVerilog applications as a minimum requirement to support the integration of Foreign Language Code. Figure 1, “Inclusion of Object Code into a SystemVerilog Application”, depicts the inclusion of object code and its relations

.

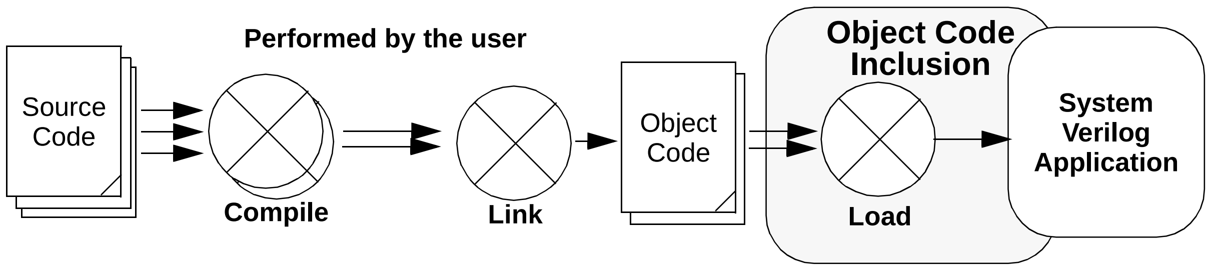

Figure 0-1 Inclusion of Object Code into a SystemVerilog Application

Compiled object code to be loaded can be specified by one of the following three methods:

1. By an entry in a bootstrap file; this file and its content will be described in more detail below. Its location must be specified with one instance of the switch “-sv_liblist <pathname>“. This switch may be used multiple times to define the usage of multiple bootstrap files.
2. By specifying the file with one instance of the switch “-sv_lib <path+name_without_extension>”; the filename must be specified without the platform specific extension. The SystemVerilog application is responsible for appending the appropriate extension for the actual platform.
   This switch may be used multiple times to define multiple libraries holding object code.
3. By including the file path and name without the platform specific extension in a list of entries defined within the environment variable SV_LIBRARIES; this variable uses the same notation and syntax as the corresponding variables[1]

All these methods must be provided and must be made available concurrently, to permit any mixture of their usage. Every location can be either an absolute pathname or a relative pathname, where the content of the environment variable SV_ROOT or a switch -sv_root is used to identify an appropriate prefix for relative pathnames.[2]

Compiled object code must be provided in form of a shared library or as an archive library having the appropriate extension for the actual platform[3]. When there exists a shared library and an archive library in the same directory, the shared library is used; otherwise the directory search order decides upon the library to be loaded.

In case of multiple occurances of a file with the same name the above order also identifies the precedence of the search; as a result a file located by method 2) will override files specified by method 3). Any library must and will only be loaded once.

1.E.g. LD_LIBRARY_PATH (on Solaris) and LPATH (on HP-UX)
2.Refer to the corresponding rules in the introduction for more details on forming pathnames.
3.Shared libraries use e.g. .so for Solaris, .sl for HP-UX and .dll for Windows, archives use .a on UNIX and .lib on Windows. In any case, it is the task of the SystemVerilog application to identify the appropriate extension.

All compiled object code must be loaded in specification order similarly to the above scheme; first the content of the bootstrap file is processed starting with the first line, then the set of '-sv_lib' switches is processed in order of their occurance, finally the content of the SV_LIBRARIES environment variable is processed from left to right.

The object code bootstrap file has the following syntax:

1. The first line must contain the string: "#!SV_LIBRARIES"
2. It follows an arbitrary amount of entries, one entry per line, where every entry holds exactly one library location. Each entry consists only of the <path+name_without_extension> of the object code file to be loaded and may be surrounded by an arbitrary number of blanks; at least one blank must preceed the entry in the line.
   The value <path+name_without_extension> is equivalent to the value of the switch '-sv_lib'.
3. Any amount of comment lines can be interspersed between the entry lines; a comment line starts with the character '#' after an arbitrary (including zero) amount of blanks and is terminated with a newline.

No other means shall be provided for identifying the location and filename of compiled object code to be included via the DirectC Interface.

Examples:

1) Assuming the environment variable SV_ROOT has been set to "/home/user" and it is needed to include the following object files

- /home/user/myclibs/lib1.so
- /home/user/myclibs/lib3.so
- /home/user/proj1/clibs/lib4.so
- /home/user/proj3/clibs/lib2.so

then this can be accomplished by one of the methods described in the following figure. Each of the four methods is equivalent.

```
#!SV_LIBRARIES
 myclibs/lib1
 myclibs/lib3
 clibs/lib4
 clibs/lib2
```

Example1a: BOOTSTRAP FILE

```
...
-sv_lib myclibs/lib1
-sv_lib myclibs/lib3
-sv_lib clibs/lib4
-sv_lib clibs/lib2
...
```

Example 1b: SWITCH LIST

```
setenv SV_LIBRARIES "myclibs/lib1:myclibs/lib3:clibs/lib4:clibs/lib2"
```

Example 1c: ENVIRONMENT VARIABLE

```
...
-sv_lib myclibs/lib1 -sv_lib clibs/lib2
...

setenv SV_LIBRARIES "myclibs/lib3:clibs/lib4"
```

Example 1d: SWITCH LIST & ENVIRONMENT VARIABLE

Please note that although Example 1d loads the same set object code as the other examples, the load order is different.

2) Assuming the environment variable SV_ROOT has been set to “/home/user” the following series of switches (left column) will result in loading the following files (right column):

`-sv_lib svLibrary1.so`	`/home/user/svLibrary1.so`
`-sv_lib svLibrary2.a`	`/home/user/svLibrary2.a`
`-sv_root /home/project2/shared_code`	
`-sv_lib svLibrary3.so`	`/home/project2/shared_code/svLibrary3.so`
`-sv_root /home/project3/code`	
`-sv_lib svLibrary4.so`	`/home/project3/code/svLibrary4.so`

Example 2: ’-sv_lib’/’-sv_root’ switches and the resulting file names