0.0.1 Inclusion of Source Code

The inclusion of source code is needed when the SystemVerilog application handles the compilation step and all subsequent activities for the user. As a result, the compilation, linking and loading steps are fully transparent to the user[1], it is only required to specify the source code files to be compiled. The inclusion of source code is intended to serve as a convinient method for including Foreign Language Code into a SystemVerilog application.

It is an optional feature that can be supported by SystemVerilog applications to support the integration of Foreign Language Code. Figure 1, “Source Code Inclusion vs. Object Code Inclusion”, depicts the inclusion of source code to the inclusion of object code.

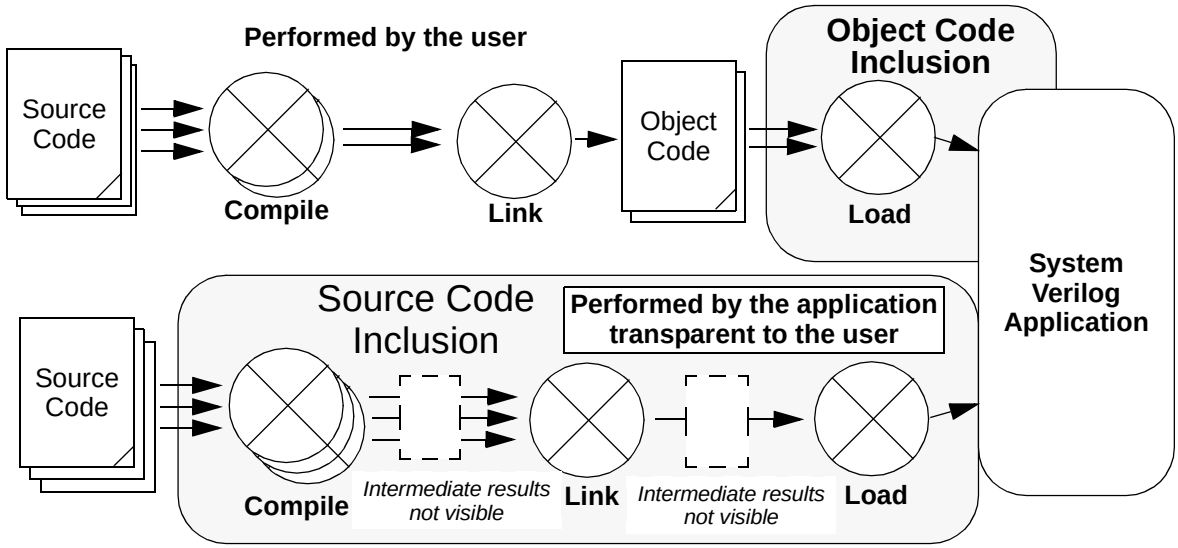

Figure 0-1 Source Code Inclusion vs. Object Code Inclusion

Similarly to the inclusion of object code, the inclusion of source code can be specified by one of the following two 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_srclist <pathname>“. This switch may be used multiple times to define the usage of multiple bootstrap files.
2. By specifying the file pathname with one instance of the switch “-sv_src <filepath>” (including the file extension). This switch may be used multiple times to define multiple source code files.

Additionally, directories holding include files needed for the compilation can be specified by:

3. By specifying one instance of the switch “-sv_inc <directorypath>”.
   This switch may be used multiple times to define multiple directories holding source code.

The specification of include file directories by an environment variable applies to all source code files, and is overridden by the switch “-sv_inc“ for all subsequent source code specifications with the switch “-sv_src“. The source code file specifications in a bootstrap file have their include directories associated within this file; and are not affected by other include directory specifications.

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 switch -sv_root is used to identify an appropriate prefix for relative pathnames.[2]

1.Otherwise the guidelines of the previous section (“Inclusion of Object Code”) would apply
2.Refer to the corresponding rules in the introduction for more details on forming pathnames.

The SystemVerilog application must use the extension of the provided source code files to distinguish between 'C' code (having the extension ".c") and source code provided in a second language (having any other extension). It must further assume that the second language is 'C++', but most other languages can be supported as well by the defined integration scheme.[1]

Dependent on the extension of the corresponding source code files, the SystemVerilog application is responsible for invoking the appropriate compiler with respect to the following scheme:

```
<compiler> <prefix_flags> <includes> <flags> <file names> <suffix_flags>
```

In the above scheme, <compiler> is a placeholder for the invocation of the appropriate compiler, which must be an ANSI-C compiler in cases where the source code file uses the extension ".c", or a C++ compiler otherwise. Likewise, <includes> is a placeholder for the list of include directories specified for the corresponding source code file. Usually this list contains zero, one or multiple entries of the form <inc_opt><include directory name> delimited by one or multiple blanks; where <inc_opt> denotes the compiler option to specify an include directory[2].
Also, <file names> is a placeholder for the specification of the input source code file and the destination file name. Usually this specification will be of the form <src_opt><source file><dst_opt><dest file>; all of these entries are delimited by one or multiple blanks. <src_opt> denotes the compiler option to specify the input (source code) file, while <dst_opt> denotes the compiler option to specify the output (destination) file. All three remaining specifications (<flags>, <prefix_flags>, and <suffix_flags>) are used to provide compilation flags; usually only <flags> will contain compilation options in a normal compilation step.

All of the above specifications can be overridden by user specific settings for both compilation modes. A set of environment variables and a corresponding set of switches has been defined for this purpose. Environment variable settings override all corresponding compilation options, while switch specifications override only subsequent source code specifications; including environment variable overrides. The following table specifies the defined set of environment variables and the corresponding switch settings.

Table 1: Environment Variables / Switches for Overriding Compilation Settings

Switch	Remarks
-sv_c_compiler <value>	Replaces the <compiler> part in case of a 'C' compilation
-sv_c_inc_opt <value>	Replaces the option to include an include directory name <inc_opt> in case of a 'C' compilation
-sv_c_src_opt <value>	Replaces the option to specify the source code file name in case of a 'C' compilation
-sv_c_dst_opt <value>	Replaces the option to specify the destination file name in case of a 'C' compilation
-sv_c_flags <value>	Replaces the <flags> part in case of a 'C' compilation
-sv_c_prefix_flags <value>	Replaces the <prefix_flags> part in case of a 'C' compilation
-sv_c_suffix_flags <value>	Replaces the <suffix_flags> part in case of a 'C' compilation
-sv_cpp_compiler <value>	Replaces the <compiler> part in case of a 'C++' compilation

1.The foreign language interface requires 'C' linkage, but is itself compiler and language independent.
2.While this option is named '-I' for many compilers, it may be different for other compilers. Please note that the SystemVerilog application must place no blank between this option name and the include directory. When a user supplied option requires a blank, then this must be inserted as part of the option name; by enclosing the option name in double quotes like a string.

Table 1: Environment Variables / Switches for Overriding Compilation Settings

Switch	Remarks
-sv_cpp_inc_opt <value>	Replaces the option to include an include directory name <inc_opt> in case of a 'C++' compilation
-sv_cpp_src_opt <value>	Replaces the option to specify the source code file name in case of a 'C++' compilation
-sv_cpp_dst_opt <value>	Replaces the option to specify the destination file name in case of a 'C++' compilation
-sv_cpp_flags <value>	Replaces the <flags> part in case of a 'C++' compilation
-sv_cpp_prefix_flags <value>	Replaces the <prefix_flags> part in case of a 'C++' compilation
-sv_cpp_suffix_flags <value>	Replaces the <suffix_flags> part in case of a 'C++' compilation

All provided source code must be compiled 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_src' switches is processed in order of their occurance.Similar applies to the inclusion of the include directories in the compilation; the order of specification must be preserved.

The source code bootstrap file has the following syntax:

1. The first line must contain the string: "#!SV_SOURCES"
2. It follows an arbitrary amount of entries, one entry per line, where every entry holds exactly one source code file location. Each entry consists at least of the <path+name_without_extension> of the source code file to be loaded; 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_src'.

   Additionally, a list of directory pathnames (separated by blanks) may be specified after the location of the source code file, separated by colon (':')[1]. Each directory entry within this list is equivalent to the value of the switch '-sv_inc'.
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 of user specific source code and include files. There is no need to locate or identify the compiled object code created from these sources; this is under the discretion of the application, no further user interaction shall be needed to accomplish this.

Examples:

1) Assuming the need to include the following source files into a simulation:

- /home/user/mycode/model1.c
- /home/user/sysc/model3.sc
- /home/user/proj1/code/model3.cc

1.The resulting syntax of an entry is:
<source code entry> ::= <blank><pathname>[[<blank>]':'[<blank>]<directories>]
<directories> ::= <pathname>[<blank><pathname>]
<blank> ::= ' '[<blank>]

- /home/user/proj3/c_code/model4.cpp

Additionally, the include directories

- /home/user/mycode/includes
- /home/user/common/sysc
- /home/user/proj1/util
- must be referenced

The first two examples show two possible methods of specifying the source files and include directories for the source code inclusion [assuming '-sv_root' specifies /home/user]. Please note that within these examples it would possible to have a finer granularity for assigning include directories to a source code file.

```
#!SV_SOURCES
 mycode/model1.c
 sysc/model3.sc
 proj1/code/model3.cc
 proj3/c_code/model4.cpp


setenv SV_INCLUDES
"mycode/includes:common/sysc:proj1/util"
```

Example 2a: BOOTSTRAP FILE & INCLUDE ENV VAR

```
-sv_src mycode/model1.c
-sv_src sysc/model3.sc
-sv_src proj1/code/model3.cc
-sv_src proj3/c_code/model4.cpp

-sv_inc mycode/includes
-sv_inc /home/user/common/sysc
-sv_inc proj1/util
```

Example 2b: SWITCH ONLY

Please note that any combination of bootstrap file, switches and/or include directories in the SV_INCLUDES environment variable is possible. This is shown in the next example, Example 2c.

```
#!SV_SOURCES
 sysc/model3.sc
 proj3/c_code/model4.cpp

setenv SV_INCLUDES "common/sysc:proj1/util"
```

```
-sv_src mycode/model1.c
-sv_src proj3/c_code/model4.cpp

-sv_inc mycode/includes
```

Example 2c: USING A COMBINATION OF BOOTSTRAP FILE, SWITCHES & INCLUDE ENV VAR

2) It is further worth to note that the bootstrap file permits a more granular assignment of include directories to source code files, as this is shown in the next example, Example 2d.

```
#!SVC_SOURCES
 mycode/model1.c : mycode/includes proj1/util common/includes
 sysc/model3.sc : common/sysc
 proj1/code/model3.cc : common/includes
 proj3/c_code/model4.cpp : proj1/util common/includes
```

Example 2d: ASSIGNING INCLUDE DIRECTORIES TO SOURCE CODE VIA THE BOOTSTRAP FILE

This example shows the specification of a compilation of

- /home/user/mycode/model1.c (using a C compiler) with the include directories: mycode/includes, proj1/util, and common/includes
- /home/user/sysc/model3.sc (using a C++ compiler) with the include directories: common/sysc
- /home/user/proj1/code/model3.cc (using a C++ compiler) with the include directories: common/includes
- /home/user/proj3/c_code/model4.cpp (using a C++ compiler) with the include dirs: proj1/util and common/includes

3) Finally a highly customized compilation with user specific options can be accomplished by the appropriate specification of environment variables and an appropriate order of switches. Assuming the following environment variable settings and switch settings:

```
SV_ROOT = “/home/user“
SV_INCLUDES = “incl_dir“

-sv_src model_list/model1.c
-sv_inc common_inc
-sv_cpp_compiler /usr/bin/g++
-sv_cpp_prefix_flags “-O3“
-sv_src model_list/model2.cpp
-sv_src model_list/model3.c
-sv_c_compiler /usr/ccs/acc
-sv_c_prefix_flags “-g -DDEBUG“
-sv_cpp_prefix_flags “-g -DDEBUG“
-sv_root /home/projects/common
-sv_inc shared_includes
-sv_src model4.c
-sv_src model5.cpp
```

the following compilations will be performed:

- /home/user/model_list/model1.c (using the default C compiler)with the include dir:
  /home/user/incl_dir
- /home/user/model_list/model2.cpp
  (using the C++ compiler “/usr/bin/g++” and the compiler flags “-O3“) with the include dir:
  /home/usr/common_inc
- /home/user/model_list/model3.c (using the default C compiler) with the include dir:
  /home/usr/common_inc
- /home/projects/common/model4.c
  (using the default C compiler with the compiler flags “-g -DDEBUG“) with the include dir:
  /home/usr/shared_includes
- /home/projects/common/model5.cpp
  (using the C++ compiler “/usr/bin/g++” and the compiler flags “-g -DDEBUG“) with the include dir:
  /home/usr/shared_includes