0.1 Code Registration

The previous two sections define how to include Foreign Language Code into a SystemVerilog application. The integration method described in those sections are sufficient for the integration of DirectC code, the association of the SystemVerilog task or function with the corresponding Foreign Language Code function is already defined. No further user involvement is required in case of DirectC.[1]

It is further possible to use all capabilities specified in the previous two sections to include code of other interfaces (e.g. PLI,VPI). In this case there is one additional functionality required for the integration of the corresponding object code; the association of user-defined system tasks and system functions with the corresponding PLI or VPI function. This activity is further referred as code registration within this section. It is the intention of this section to specify guidelines for this purpose, with the intent to provide a common method of including PLI and VPI code that is consistent with the integration of DirectC code. This is important, since DirectC code might use PLI and/or VPI functions within its Foreign Language Code.

As such, this section builds upon the content of the previous two sections; the capabilities defined herein should be provided to also include PLI and/or VPI functions into a SystemVerilog application without any modification. The only further requirement is to associate the PLI and/or VPI functions with the appropriate system task or system function, an activity that is regarded as being completely orthogonal with loading the related object code. The corresponding definitions are made within this section.

A SystemVerilog application must provide the following capabilities to register PLI and/or VPI functions loaded by one of the methods described in the previous sections:

1. By specifying a registration function with one instance of the switch “-sv_register <funcname>”; where <funcname> specifies a registration function, which must be part of one of the loaded compiled objects. The value of <funcname> is the name of the corresponding function. The SystemVerilog application is responsible for calling this function.
   This switch may be used multiple times to define multiple registration functions.

   The signature of this function is `void funcname( void );`

   This function is intended to permit the registration of VPI functions by `vpi_register_systf()`. The SystemVerilog application should call any function specified by the switch “-sv_register“ like it has been specified as a part of the `vlog_startup_routines`.[2]
2. By specifying a file that contains all required registration information with one instance of the switch “-sv_pli_file <file>”. The expected syntax of a PLI registration file is described below. The SystemVerilog application is responsible for reading this file and registering every function identified within this file.
   This switch may be used multiple times to define multiple files holding registration information.

   The signature of this function is `void funcname( void );`
3. By specifying a registration function with one instance of the switch “-sv_pli_func <funcname>”; where <funcname> specifies a registration function, which must be included in one of the loaded compiled objects. The return value of this function is a pointer to a structure holding the registration information. The SystemVerilog application is responsible for calling this function, interpreting the content of the returned structure and for registering the corresponding functions.
   This switch may be used multiple times to define multiple libraries holding object code.

   The signature of this function is `t_tfcell* funcname( void );`

1.Therefore the content of this section relates only to PLI and VPI functions to be included into a SystemVerilog application.
2.

PLI Registration file syntax

The PLI registration file has the following syntax:

- It consists of an arbitrary amount of entries, one entry per line, where every entry holds the information to register exactly one PLI system function or task. Each entry consists of the name of the system function or task (including a leading $ character) followed by one or multiple function specifications, all delimited by one or multiple blanks. The following function specifications are possible:
  - call=function Specifies the name of the call function
  - check=function Specifies the name of the check function
  - misc=function Specifies the name of the misc function

  at least one of the above entries must be provided in a registration entry. Omitted entries result in no corresponding function being registered.
  - args=<number> Specifies the number of arguments to the user-defined system task or system function
  - minargs=<number> Specifies the minimum number of arguments
  - maxargs=<number> Specifies the maximum number of arguments
  - data=<number> Specifies the data value passed as first argument to the call, check, and misc function. The default value is 0, this value is used when this specification is omitted. This argument can be used to distinguish different calls, when multiple SystemVerilog system tasks or functions are mapped to a single PLI function.
  - size=<number> Specifies the size of the returned value in bits. This specification is mandatory for user-defined system functions. Specify 0 or omit this specification in case of a user-defined system task.
  - persistent Enables to enter the user-defined system task on the command line

  All of the above specification entries must be supported by a SystemVerilog application.

  Additional, vendor specific function specifications are possible and permitted and must be ignored when not understood by the actual SystemVerilog application. All vendor specific function specifications must use the following syntax <funcspec_name>=<funcspec_value(s)> or <funcspec_name>.

register_entry ::= systemtaskfunctionname blanks function_specs
systemtaskfunctionname ::= “$“string[a]
function_specs ::= [opt_specs blanks] name_specs [blanks opt_specs]
name_specs ::= name_spec name_specs
opt_specs ::= arg_spec | other_spec | vendor_spec blanks opt_specs
name_spec ::= “call=“function | “check=“function | “misc=“function
arg_spec ::= “args=”number | “minargs=“number | “maxarg=“number
other_spec ::= “data=”number | “size=“number | “persistent”
vendor_spec ::= string[”=”string]

Table 1: BNF for the PLI registration file syntax

a.function denotes the name of a C function, string denotes a string not enclosed by double quotes, while number denotes a positive integer number. blanks denotes one or multiple blanks.

- 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.

PLI Registration Structure

The PLI registration structure expected from a function used by method 2) is defined as follows:

```
s_tfcell return_value[] = {
        { usertask, 0, my_check, 0, my_call, my_misc, “$my_task“ },
        { usertask, 0, 0, 0, abc_calltf, 0, "$abc" },
        {0}   /* final entry must be 0 */
};
```

The definition of a t_tfcell entry is as follows:

```
typedef int (*p_tffn) ();

typedef struct t_tfcell {
    short type;   /* one of the keywords: usertask, userfunction, userrealfunction */
   short data;   /* Specifies the data value passed as first argument to the call, check, and misc function. */
    p_tffn checktf; /* Specifies the name of the check function, null means no function */
   p_tffn sizetf;   /* Returns the size in bits of the value returned a system function , null means no function */
    p_tffn calltf;   /* Specifies the name of the call function, null means no function */
    p_tffn misctf;   /* Specifies the name of the misc function, null means no function */
    char* tfname;   /* Name of the system function or task including leading $ */
    int forwref;            /* always set to 1*/
    char *tfveritool;   /* For compatibility reasons, do not alter this field */
    char *tferrmessage;    /* For compatibility reasons, do not alter this field */
    int hash;    /* For compatibility reasons, do not alter this field */
    struct t_tfcell *left_p;    /* For compatibility reasons, do not alter this field */
    struct t_tfcell *right_p;     /* For compatibility reasons, do not alter this field */
    char *namecell_p;    /* For compatibility reasons, do not alter this field */
    int warning_printed;    /* For compatibility reasons, do not alter this field */
} s_tfcell, *p_tfcell;
```

Table 2:

The storage for each t_tfcell entry passed to the simulator must persist throughout the simulation because the simulator might de-references the pointer to call the callback functions. Also, the last entry must be set to 0.

All registration functions must be called in order of their specification on the SystemVerilog application invocation.

Example:

<missing>