

20. PLI overview

20.1 PLI purpose and history (informative)

The 1364-???? specification has deprecated the TF and ACC routines which were specified previously in [Clause 20](#) through [Clause 25](#) and [Annex E](#) and [Annex F](#). [Clause 20](#) has been modified to reflect this change. The other Clauses and Annexes have been eliminated entirely.

[Clause 26](#) and [Clause 27](#) and [Annex G](#) describe the C language procedural interface standard and interface mechanisms that are part of the Verilog HDL. This procedural interface, known as the Programming Language Interface, or PLI, provides a means for Verilog HDL users to access and modify data in an instantiated Verilog HDL data structure dynamically. An instantiated Verilog HDL data structure is the result of compiling Verilog HDL source descriptions and generating the hierarchy modeled by module instances, primitive instances, and other Verilog HDL constructs that represent scope. The PLI procedural interface provides a library of C language functions that can directly access data within an instantiated Verilog HDL data structure.

A few of the many possible applications for the PLI procedural interface are:

- C language delay calculators for Verilog model libraries that can dynamically scan the data structure of a Verilog software product and then dynamically modify the delays of each instance of models from the library
- C language applications that dynamically read test vectors or other data from a file and pass the data into a Verilog software product
- Custom graphical waveform and debugging environments for Verilog software products
- Source code decompilers that can generate Verilog HDL source code from the compiled data structure of a Verilog software product
- Simulation models written in the C language and dynamically linked into Verilog HDL simulations
- Interfaces to actual hardware, such as a hardware modeler, that dynamically interact with simulations

This document standardizes the Verilog PLI that has been in use since 1995. The PLI comprises three primary generations of the Verilog PLI.

- a) *Task/function* routines, called *TF* routines, made up the first generation of the PLI. These routines, most of which started with the characters **tf_**, were primarily used for operations involving user-defined task/function arguments, along with utility functions, such as setting up call-back mechanisms and writing data to output devices. The TF routines were sometimes referred to as *utility* routines

Note: The TF routines have been deprecated. Clauses 21, 24 and 25 and Annex F have been removed as a result.

- b) *Access* routines, called *ACC* routines, formed the second generation of the PLI. These routines, which all started with the characters **acc_**, provided an object-oriented access directly into a Verilog HDL structural description. ACC routines were used to access and modify information, such as delay values and logic values on a wide variety of objects that existed in a Verilog HDL description. There was some overlap in functionality between ACC routines and TF routines.

Note: The ACC routines have been deprecated. Clauses 21 through 23 and Annex E have been removed as a result.

- c) *Verilog Procedural Interface* routines, called *VPI* routines, are the third generation of the PLI. These routines, all of which start with the characters **vpi_**, provide an object-oriented access for both Verilog HDL structural and behavioral objects. The VPI routines are a superset of the functionality of the TF routines and ACC routines.

20.2 User-defined system task or function names

A user-defined system task or function name is the name that will be used within a Verilog HDL source file to invoke specific PLI applications. The name shall adhere to the following rules:

- The first character of the name shall be the dollar sign character (\$)
- The remaining characters shall be letters, digits, the underscore character (_) or the dollar character (\$)
- Uppercase and lowercase letters shall be considered to be unique—the name is case sensitive
- The name can be any size, and all characters are significant

20.3 User-defined system task or function types

The type of a user-defined system task or function determines how a PLI application is called from the Verilog HDL source code. The types are:

- A user *task* can be used in the same places a Verilog HDL task can be used (refer to [10.2](#)). A user-defined system task can read and modify the arguments of the task, but does not return any value.
- A user *function* can be used in the same places a Verilog HDL function can be used (refer to [10.3](#)). A user-defined system function can read and modify the arguments of the function, and shall return a scalar or vector value. The bit width of the return value shall be determined by a user-supplied *size* application (see [21.1.1](#) **This reference should be to 27.34**).
- A user *real-function* can be used in the same places a Verilog HDL function can be used (refer to [10.3](#)). A user-defined system real-function can read and modify the arguments of the function, and will return a double-precision floating point value.

20.4 Overriding built-in system task and function names

[Clause 17](#) defines a number of built-in system tasks and functions that are part of the Verilog language. In addition, software products can include other built-in system tasks and functions specific to the product. These built-in system task and function names begin with the dollar sign character (\$) just as user-defined system task and function names.

If a user-provided PLI application is associated with the same name as a built-in system task or function (using the PLI interface mechanism), the user-provided C application shall override the built-in system task/function, replacing its functionality with that of the user-provided C application. For example, a user could write a random number generator as a PLI application and then associate the application with the name **\$random**, thereby overriding the built-in **\$random** function with the user's application.

Verilog timing checks, such as **\$setup**, are not system tasks, and cannot be overridden.

The system functions **\$signed** and **\$unsigned** can be overridden. These system functions are unique in the Verilog HDL, in that the return width is based on the width of their argument. If overridden, the PLI version shall have the same return width for all instances of the system function. The PLI return width is defined by the PLI *size* routine.

20.5 User-supplied PLI applications

User-supplied PLI applications are C language functions that utilize the library of PLI C functions to access and interact dynamically with Verilog HDL software implementations as the Verilog HDL source code is executed.

These PLI applications are not independent C programs. They are C functions, which are linked into a software product, and become part of the product. This allows the PLI application to be called when the user-

defined system task or function \$ name is compiled or executed in the Verilog HDL source code.

20.6 PLI interface mechanism

The PLI interface mechanism provides a means to have PLI applications called for various reasons when the associated system task or function \$ name is encountered in the Verilog HDL source description. For example, when a Verilog HDL simulator first compiles the Verilog HDL source description, a specific PLI routine can be called that performs syntax checking to ensure the user-defined system task or function is being used correctly. Then, as simulation is executing, a different PLI routine can be called to perform the operations required by the PLI application. Other PLI routines can be automatically called by the simulator for miscellaneous reasons, such as the end of a simulation time step or a logic value change on a specific signal.

20.7 User-defined system task and function arguments

When a user-defined system task or function is used in a Verilog HDL source file, it can have arguments that can be used by the PLI applications associated with the system task or function. In the following example, the user-defined system task \$get_vector has two arguments:

```
$get_vector("test_vector.pat", input_bus);
```

The arguments to a system task or function are referred to as *task/function arguments* (often abbreviated as *tfargs*). These arguments are not the same as C language arguments. When the PLI applications associated with a user-defined system task or function are called, the task/function arguments are not passed to the PLI application. Instead, a number of PLI routines are provided that allow the PLI applications to read and write to the task/function arguments. Refer to [Clause 27](#) for information on specific routines that work with task/function arguments.

20.8 PLI include file

The libraries of PLI functions are defined in a C include file, which is a normative part of the 1364 standard. This file also defines constants, structures, and other data used by the library of PLI routines and the interface mechanisms. The file is `vpi_user.h` (listed in [Annex G](#)).

- PLI applications that use the VPI routines shall include the file `vpi_user.h`.