Using PLI 2.0 (VPI) with VCS
(Yes, it really works!)

Stuart Sutherland

Sutherland HDL, Inc., Portland, Oregon
stuart@sutherland-hdl.com

ABSTRACT

The Verilog PLI VPI library, often referred to as “PLI 2.0”, is the latest generation of the Verilog PLI standard. The VPI library has a number of advantages over the older TF and ACC libraries (often collectively referred to as “PLI 1.0”). The VPI library has been part of the IEEE 1364 Verilog standard since 1995, but the VCS simulator has only just begun to support the VPI library with version 6.1, which is projected to be released early in 2002. This paper presents why it is desirable to use the VPI library, and how well the library is supported in VCS. The goal of the paper is to answer the question: *“Are there compelling reasons to use PLI 2.0 in your future PLI applications?”*

1.0 What are “PLI 1.0” and “PLI 2.0”

The PLI has been an integral part of the Verilog language since 1985, and has been a major contributor to the success of Verilog. There have been a number of versions of the PLI in its 17-year history.

1985: The TF library. The original Verilog PLI was designed to read and write the arguments of a system task. For the most part, this first generation of the PLI could not see the internals of a simulation data structure; it could only see the information passed in as system task arguments. This first generation of the PLI is a library of C functions which mostly begin with the letters *“tf_”* (for “task/function”). Hence, the library is often referred to as the “TF” library. In most Verilog simulators, the TF library is defined in a file called *veriuser.c*. Until version 6.1, however, VCS placed the library in a file called *vcs_user.c*. VCS 6.1 now uses the standard veriuser.c file.

1988: The ACC library. A second generation of the PLI was created to supplement and extend the capability of the TF library. This second library is defined in a different C file, called acc_user.h. All the routines in this library begin with the letters *“acc_”*, which stands for *“access”*. The ACC library was developed specifically at the request of ASIC vendors to do delay calculation, power analysis and other types of analysis involving the cells that make up an ASIC netlist. The ACC library provides access into the simulation for structural designs (netlists). The ability to access the structural level of a design has made the ACC library very valuable for a wide variety of other types of applications as well. Waveform displays such as VirSim and other graphical design debug utilities often use the ACC library.

A primary limitation of the ACC library, however, is that it can only access the structural level of Verilog models. The ACC library cannot access RTL models, memory arrays and many other types of objects that make up a large part of many Verilog HDL models. Another drawback of the ACC library is its ad-hoc evolution. Much of the library of C functions were added after the original release in 1988. These additional routines were added a little here and a little there, with no guiding specification to ensure consistency in syntax and semantics. As a result, the ACC library is full of inconsistencies, redundancies and poorly defined behavior. It is awkward to learn and difficult for simulation vendors to implement. The ACC library behaves differently in every Verilog simulator.

1990: The OVI “PLI 1.0” standard. When Gateway Design Automation/Cadence released the Verilog language and PLI to the public domain, the users forum *Open Verilog International* (OVI) labeled the combined TF/ACC libraries as *“PLI 1.0”*. This term was strictly a label—PLI 1.0 did not add any new features or capabilities to the Verilog PLI.

1993: The OVI “PLI 2.0” standard. OVI decided to completely replace the TF and ACC libraries with a new library, that eliminated the redundancies and limitations of the old libraries. OVI called the new library *“PLI 2.0”*. This new library combined all the functionality of the 200+ routines in the TF and ACC libraries into a single library of about 25 routines. The syntax of these routines were simple and well defined. PLI 2.0 overcame many, if not all, of the weaknesses and drawbacks of the ACC library.

OVI intended to totally eliminate the old TF and ACC libraries, and therefore deliberately made PLI 2.0 so that it was *not* backward compatible. Indeed, OVI’s PLI 2.0 called its library ACC, and used many of the same function names and constant names as the older ACC library, but with dif-

ferent functionality and values. Because PLI 2.0 was not backward compatible, the hundreds of existing PLI applications could not be used together with a PLI 2.0 application. Therefore, no simulators ever completed implementing the OVI version of the PLI 2.0 standard. The original OVI "PLI 2.0" never saw the light of day.

1995: The IEEE 1364-1995 standard[1] and the VPI library. In 1995, the IEEE standardized both the Verilog HDL and PLI the way it stood in 1993. No enhancements were considered for 1364-1995. For the PLI, the IEEE chose to standardize both PLI 1.0 and 2.0. The former, to preserve backward compatibility. The latter, because it really was a much better procedural interface. To allow both the old PLI 1.0 and the incompatible PLI 2.0 standards to be used at the same time, the IEEE rewrote the PLI 2.0 so that it was backward compatible. As part of that process, the PLI 2.0 routines were renamed to *"VPI"* (for *"Verilog Procedural Interface"*).

Note: *In the IEEE standard, the terms "PLI 1.0" and "PLI 2.0" do not exist. There is one PLI, with three libraries, TF, ACC and VPI.*

2001: The IEEE 1364-2001 standard[2]. The IEEE spent three years defining a major set of enhancements to the Verilog standard. These enhancements were ratified early in 2001. The new Verilog-2001 standard adds a number of powerful enhancements to the Verilog HDL, including: multi-dimensional arrays, re-entrant tasks, recursive functions, configurations, attributes (which can replace those annoying synthesis pragmas hidden in comments), and several dozen other useful and important features. For more details on these features, refer to the SNUG San Jose 2001 Conference paper, *"Getting the Most out of IEEE 1364-2000 Verilog Standard"*[3].

2.0 Why use the VPI library?

To understand the advantages gained by using the VPI library, we must first look at the strengths and weaknesses of the older TF and ACC libraries.

2.1 Strengths and weaknesses of the TF library

The TF library can only access the arguments of a system task or function. Due to this limited access, simulators can predict at compile time exactly what information a PLI application will access. This allows simulators to highly optimize the simulation data structure. VCS takes full advantage of this, and achieves its best simulation performance when only the TF library is used. However, the TF library cannot traverse design hierarchy, analyze design structure, modify delays, or access RTL code. This greatly limits the types of PLI applications that can be written with the TF library. Originally, the TF library contained just a few C functions. Over a number of years, however, the library evolved until it grew to have more than 110 C functions. This growth occurred with no specification to guide the evolution. As a result, the library is full of inconsistencies and redundancies. The TF library also has a great number of problems with portability to different simulators and different operating systems. Hence PLI applications do not work the same way on all simulators. In part, this portability problem stems from the fact that the TF library was originally designed to work with just the Cadence Verilog-XL simulator, in the days of DEC PDP-11 computers. The TF library does not support many of the new constructs added in the IEEE 1364-2001 standard.

2.2 Strengths and weaknesses of the ACC library

The ACC library is an extension to the TF library. It provides access to structural objects within a simulation data structure. Structural objects include module instances, primitive instances, delays, net declarations, and variable declarations. The access to structural objects is achieved using routines which can search for objects. This overcomes the primary limitation of the TF library, which requires objects to be listed as system task/function arguments. However, the ACC library suffers from many of the same problems as the TF library. There are well over 100 C functions, with inconsistent syntax and semantics. The specification and documentation of the ACC library routines were grossly lacking when the library was first placed in the public domain. This has led to differences in the way the ACC routines work on different simulators.

The ACC library can arbitrarily access any structural object anywhere within the simulation data structure. This provides the capabilities needed for SDF back annotation, power analysis, gate-level logic flow tracing, waveform displays, and a wide variety of commercial and in-house PLI applications. But, the ACC library is limited to only accessing structural (netlist) based designs. The ACC library cannot access all types of objects that make up the typical Verilog design. Verilog procedures, continuous assignments and memory arrays and other components of an RTL level model are *not* accessible by the ACC library, and it cannot access much of what is in a typical test bench.

Performance is another limitation of the ACC library. The arbitrary access to structural objects within the simulation data structure means that a simulation compiler cannot predict what will be accessed, and therefore cannot optimize the data structure as effectively. The run-time performance of VCS slows down dramatically if the ACC library is granted full access to the entire simulation data structure.

The ACC library cannot access many of the new constructs in the IEEE 1364-2001 standard.

2.3 Strengths and weaknesses of the VPI library

The VPI library is a complete superset of the older TF and ACC libraries. It provides the advantages of both libraries, yet strives to eliminate the many disadvantages. The primary advantages are:

- The VPI library replaces over 220 C functions that are complex, inconsistent and have portability problems, with 37 C functions that have a simple, consistent syntax.
- The VPI library provides full access to the RTL and behavioral code within a simulation data structure.
- The access methods used by the VPI library to find information in the simulation data structure are architected in such a way that they can be implemented more efficiently in simulators, (though there is nothing in the standard to compel a simulator to be more efficient).
- The VPI library encourages a more disciplined structured programming style. The way the VPI library locates objects within the simulation data structure also encourages writing code that is efficient for simulation performance and memory usage. This is very different than the TF and ACC libraries, which encourage a very unstructured and highly inefficient programming style.

- Verilog-2001 adds over 45 major enhancements to the Verilog HDL. The VPI library was updated to provide full access to all these new capabilities.
- The VPI library was designed to be able to grow with the Verilog language. When the next generation of the IEEE Verilog standard comes out, the VPI library will be able support whatever new capabilities are added to the language.

The last two advantages are critical advantages of the VPI library. The size of designs that engineers are modeling is increasing rapidly. The software tools engineers use must evolve, in order to be useful on future designs. Verilog has evolved, and will continue to evolve. The IEEE 1364 standards group that oversees the Verilog language long ago determined that it was not logical—and even impossible—to keep adding to the bloated TF and ACC libraries, with their inconsistencies and compatibility problems. The IEEE standards group is only evolving the VPI library. *A PLI application will only work with the new Verilog-2001 features and whatever comes after Verilog-2001, if the PLI application is written with the VPI library.*

3.0 The nuts and bolts of the VPI

The following subsections explain the basics of how the VPI library works. The purpose of this explanation is to provide the background necessary to answer the question as to whether the VPI offers advantages that are compelling enough to use it with VCS.

3.1 The VPI Library

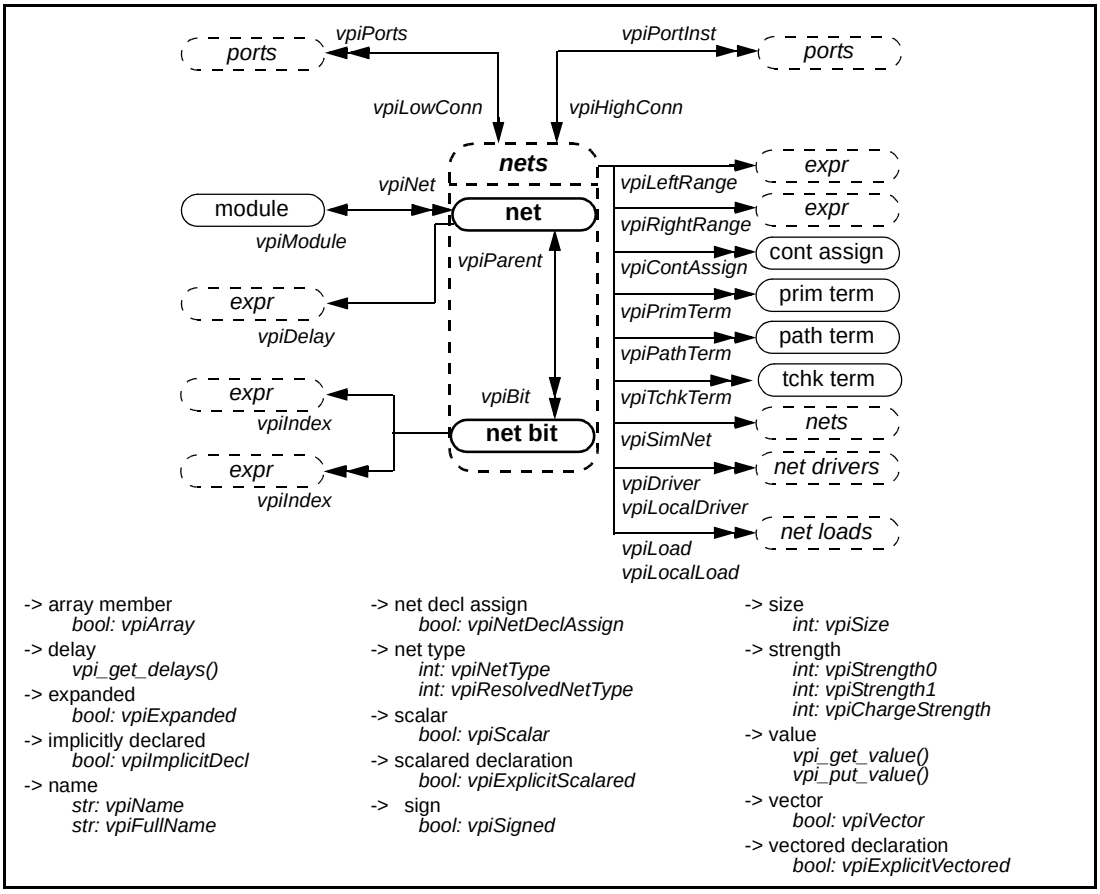
The Verilog-2001 VPI library comprises 37 C functions, with accompanying C structures and constant definitions. In general, the functions are divided into three categories: those that locate objects, those that read and modify information about objects, and utility routines for tasks such as controlling simulation and file I/O.

3.2 Object Handles

The VPI works by obtaining a “handle” to an object. A handle is a pointer to a structure containing information about the object. In this regard, the VPI works similarly to the ACC routines. However, the VPI considers virtually every thing that exists in a Verilog model, or within a Verilog simulation, as an object. The ACC library only considers the structural items of a design as models, such as nets, primitive instances and module instances.

3.3 Object Diagrams

The IEEE VPI standard provides detailed diagrams showing the relation of each object to other objects. For example, the diagram for a net object shows all the types of Verilog constructs a net could be connected to. Using the diagrams, a PLI application developer can easily determine how to obtain a handle to any specific object. The diagrams also document what information is available about each object, such as its name, data type or logic value. The figure below illustrates a sample diagram.



In the preceding diagram, the objects in **bolded** text are the ones being defined in the diagram. The objects in non-bolded text have a relationship to the defined object. A single arrow indicates there is a one-to-one relationship, and a double arrow indicates there is a one-to-many relationship. A net is declared within a single module, for example, so there is a single arrow going from **net** to module. On the other hand, a module can contain many nets, so there is a double arrow going from module to **net**. Below the diagram, all the properties for the defined objects are listed, along with the C constant or VPI routine used to access that property. For example, the diagram shows that a net has a name and a size property.

There is a similar VPI object diagram for each and every object that exists within the Verilog simulation data structure. These diagrams clearly define the semantics of the VPI library, so that all simulators can implement the VPI library exactly the same way.

3.4 Obtaining Handles

The VPI library uses three basic routines to obtain object handles: `vpi_handle()`, `vpi_iterate()` and `vpi_scan()`. These three VPI routines replace more than 80 routines in the old ACC library.

3.5 Reading Information

The VPI library uses two basic routines to read information about objects, `vpi_get()` and `vpi_get_str()`. These routines replace more than 50 routines in the old TF and ACC library.

3.6 Changing Values

The VPI library uses two basic routines to read and change values, `vpi_get_value()` and `vpi_put_value()`. These routines use a consistent syntax, no matter what type of object is be read or written, and no matter how the value is represented in C (int, double, string, etc.).

3.7 Simulation Callbacks

The VPI library can “register” with a Verilog simulator to call a VPI-based PLI application for a wide variety of events and actions that may occur during simulation. Some, such as for a value change on a net, could also be done using the older TF and ACC libraries. However, the VPI offers many types of callbacks that could not be done with the older libraries, such as monitoring which branch of an `if–else` branch or `case` statement branch is taken.

3.8 An example of a VPI-based PLI application

The example shown below is purposely short and simple, but illustrates what a VPI-based PLI application looks like. This example is called from a Verilog test bench, as follows:

```
module test;
  my_chip u1;
  initial
     $list_nets(test.u1);
endmodule
```

The example reads a module instance name from a system task argument, and then locates and prints the names of all nets within the module.

```
#include <stdio.h>      /* standard C I/O library */
#include “vpi_user.h”   /* the VPI library is defined in this file */
int ListNetsCall(char *user_data)
{
  vpiHandle systf_handle, arg_iterator, module_handle, net_iterator, net_handle;

  /* get a handle to the first argument to $list_nets */
  systf_handle = vpi_handle(vpiSysTfCall, NULL);
  arg_iterator = vpi_iterate(vpiArgument, systf_handle);
  module_handle = vpi_scan(arg_iterator);
  vpi_free_object(arg_iterator);

  /* get the name of the module represented by that first argument, and print it */
  vpi_printf(“\nModule %s contains nets:\n”,
               vpi_get_str(vpiDefName, module_handle));

  /* search for all nets within the module, and print the name of each net found */
  net_iterator = vpi_iterate(vpiNet, module_handle);
  if (!net_iterator)
     vpi_printf(“   no nets found.\n”);
```

```
    else
       while (net_handle = vpi_scan(net_iterator))
          vpi_printf("   %s\n", vpi_get_str(vpiName, net_handle));

    return(0);
  }
```

4.0 Fastening the nuts onto the bolts

PLI applications consist of one or more C functions. Each function is classified as:

- A *checktf routine* (called a *compiletf routine* in the VPI library), which is executed before simulation starts running, and performs syntax checking on the usage of the system task or system function within the Verilog HDL code.
- A *calltf routine*, which is invoked by the simulator every time simulation encounters the system task or system function within the Verilog HDL code execution. In the following example, the calltf routine for $get_vector would be executed at every positive edge of the master_clock:

```
always @(posedge master_clock)
  $get_vector("vector_file.dat", data_bus);
```

- A *misctf routine*, which is invoked by the simulation for miscellaneous events, such as when a breakpoint is encountered in the Verilog HDL source code, or when simulation exits.
- A *sizetf routine*, which is called during the simulation elaboration phase to determine the number of bits returned by a system function (Note: VCS has never supported the sizetf routine, though there are plans to add this support in a future release).

The IEEE Verilog standard defines two "interface mechanisms" to inform Verilog simulators about PLI applications: one for applications which use the older TF and ACC libraries, and a newer interface for applications which use the VPI library. In order to explain how VPI-based PLI applications are linked into the VCS simulator, it is necessary to understand the fundamentals of the older interface as well.

4.1 The old TF/ACC interface, and its limitations

The IEEE 1364 standard is deliberately vague about how PLI applications based on the TF/ACC libraries should be "interfaced" into a Verilog simulator. This is because the standard was not created until after the PLI had been in the public domain for many years, and different simulators had created different interfaces. To maintain backward compatibility, the IEEE standard allows all existing interfaces.

The one consistent facet of all simulators is that very detailed information about the PLI application must be given to the end user. In order to interface a TF/ACC based application, the user must know: the system task/function name, a static user-data value, the name of the internal C function for the calltf routine, the name of the internal C function for the checktf routine, the name of the internal C function for the misctf routine, the name of the internal C function for the sizetf routine, the type of value a system function returns, and the vector size a system function returns. Obtaining all this information puts a large burden on the end-user of the PLI application.

The VCS method of interfacing TF/ACC based PLI applications is one of the simplest in the industry. The information about the PLI application is specified in a PLI table file, often called a "tab" file, because the file extension is .tab. The name of this file is passed to the VCS compiler using the -P invocation option. An example of two entries in a PLI table file are:

```
$list_nets check=check_args call=listnets misc=cleanup data=0
$get_value check=getval_check call=getval_call size=64 data=0 +acc=read:top.i1+
```

In addition to specifying the PLI interface information, the VCS PLI table file also specifies the access that ACC routines will be allowed. By default, the access is severely restricted, which allows the VCS compiler to perform the greatest amount of performance optimizations. Access can be turned on for specific regions of the design. Access can also be enabled just for specific types of ACC routines. By restricting ACC routine access, simulation run-time performance can be dramatically improved. This concept is discussed further in section 6.1, under "performance".

Important disadvantage: The old TF/ACC interface mechanism is error prone. If the end-user of a PLI application does not know all the internal information of the application—or if the end user inadvertently specifies the information incorrectly—then the PLI application will not work properly. The user's error might not be detected until many hours into a simulation. This disadvantage of the old TF/ACC interface exists for all Verilog simulators

4.2 The IEEE 1364 VPI standard interface

The IEEE 1364 Verilog standard VPI interface eliminates all of the shortcomings of the old TF/ACC interface. The standard very explicitly defines how VPI based applications should be interfaced with Verilog simulators. This means every IEEE *compliant* Verilog simulator will use the same, consistent method. The standard VPI interface is very simple. Within the PLI application, a "register function" is defined. This function specifies the system task/function name, the system function return type, a dynamic user-data value, and the names of the internal functions that make up the calltf, compiletf and sizetf routines.

Important advantage: The VPI interface information is coded into the PLI application. The end-user of an application does not need to know—and cannot mess up—the detailed internal information about the application.

Another important feature of the VPI interface is that the user-data value can be dynamically calculated by the PLI application. For example, the user-data value can be used to store a pointer to dynamically allocated memory that will be used by the PLI application. This capability did not exist in the old TF/ACC interface, where the user-data value was a simple static integer, that had to be specified by the end user of the application.

5.0 Does the VPI actually work with VCS?

VCS 6.1 supports a substantial, and useful, portion of the VPI standard, as defined in the IEEE 1364-2001 "Verilog-2001" standard.

5.1 Enabling VPI support

In VCS 6.1, support for the VPI library must be enabled using a **+vpi** compile-time option. For example:

```
vcs +vpi test.v -P test.tab test.c
```

VCS 6.1 also uses the same ACC access commands in the PLI table file to specify the allowable access that VPI routines will have in the simulation data structure. The more restrictive the access, the more the compiler can do performance optimizations. The author of this paper strongly endorses this approach to optimize simulation performance. It allows PLI applications to access whatever information is required, without having to pay a performance penalty—which can be substantial—for retaining information that the PLI application does not require.

5.2 Unsupported VPI capabilities

VCS 6.1 is the first release to support any of the VPI library. In this release, nearly all of the routines in the VPI library for the Verilog-2001 standard are supported. A few of these VPI routines are not 100% complete in VCS 6.1. Synopsys engineering will be adding support for most of the remaining VPI capabilities during 2002. *The VCS engineering team would like to hear from you, their customers, as to which capabilities should have the highest priority.*

The following table lists the few VPI features which are not fully implemented in VCS 6.1. This information was taken from the release notes for VCS 6.1 beta[4].

`vpi_register_systf()`	not supported in VCS 6.1
`vpi_sim_control()`	not supported in VCS 6.1
`vpi_put_data()` `vpi_get_data()`	not supported in VCS 6.1
`vpi_get_delays()` `vpi_put_delays`	VCS 6.1 does not support continuously assigned objects and procedurally assigned objects.
`vpi_get_value()`	VCS 6.1 does not support: variable selects, primitives, function calls, system function calls, operations, attributes, UDP table entries with a `vpiVectorVal` format.
`vpi_put_value()`	VCS 6.1 does not support: variable selects, primitives, function calls, attributes; `vpiStrengthVal` format is only partially supported
`vpi_register_cb()`	VCS 6.1 has not implemented the callbacks: `cbEndOfSimulation`, `cbError`, `cbPliError`, `cbSignal`, `cbForce`, `cbRelease`, `cbAssign`, `cbDeassign`, `cbTchkViolation`; `cbValueChange` is implemented for all but arrays

The subset of the VPI library which is supported in VCS 6.1 is very useful. A large variety of PLI applications can be written with this subset. The book *"Verilog PLI Handbook, at Tutorial and*

Reference on the Verilog PLI"[5] contains approximately 30 examples of VPI-based PLI applications. The majority of these applications run with VCS 6.1.

5.3 The VCS 6.1 VPI interface

VCS 6.1 has not implemented the IEEE standard VPI interface. Instead, VCS 6.1 uses its older TF/ACC interface to register VPI applications. That is, the name of the system task or system function and the names of the compiletf and calltf routines are all specified in a VCS PLI table file. The VCS PLI table file does not support the sizetf routine. Instead, the size and type of system function return values must be hard coded into the table file. Synopsys does have plans to add support for the sizetf routines in a future version of VCS.

Using the old PLI 1.0 interface once again puts the burden on the end-user of knowing and correctly specifying the internal information about the application. Thus, VCS 6.1 does not take advantage of one of the major advantages of the standard VPI interface. Using the old PLI 1.0 interface also limits some of the capabilities of the VPI. Some of the VPI functionality that will not work in the VCS proprietary VPI interface are:

- Dynamically calculated information cannot be stored in the user-data field.
- The return size of system functions cannot be calculated by the PLI application.
- PLI applications cannot be registered to be called at the very start of simulation, before any simulation events have been executed.

There is a work around to the last limitation, but it requires modifying PLI applications which were written using the IEEE standard interface. The compiletf routine is supported by VCS. This routine is called prior to simulation time zero, and can be used to register other PLI applications that need to be called at the very start of simulation, before any simulation events have been executed.

The author of this paper considers VCS's proprietary VPI interface a serious limitation of VCS 6.1, and hopes that future releases of VCS will support the IEEE standard VPI interface.

6.0 Do you really want to switch to using the VPI?

The focus of this paper has been on the strengths and weaknesses of the VPI library. The purpose for this has been to answer the question: ***"Are there compelling reasons to make you want to use the VPI for your future PLI applications?"***

The answer is, ***"Yes, you do want to write most future PLI applications using the VPI routines!"***

The VPI provides access to the RTL and behavioral levels of the Verilog HDL. This extended access opens the door to many new possible PLI applications, which could not be done with the older TF and ACC libraries.

The VPI library, and how it accesses objects and information, is very thoroughly documented in the IEEE 1364 Verilog standard. The old, bloated TF and ACC libraries evolved before there was

a Verilog standard, and are full of ambiguities, inconsistencies and portability issues. Simulator companies can implement the VPI library without this ambiguity. The well defined VPI library leads to PLI applications that port well to all IEEE compliant simulators.

The VPI library is architected in such a way that it requires the PLI application developer to use a more structured programming style. This encourages—almost forces—the application developer to write code that utilizes memory more efficiently, and that executes more efficiently. Conversely, the TF and ACC libraries encourage—almost force—an unstructured, very inefficient coding style. PLI applications developed using the VPI library will most likely execute much faster and require less memory.

Perhaps the most compelling reason for using the VPI library in PLI applications is support for the Verilog-2001 standard. The VPI is the future of the PLI! In order for a PLI application to work with the new features of Verilog-2001 and any future Verilog standards, the application *must* be written using the VPI library.

6.1 Are there any reasons not to use the VPI?

There are three common reasons many companies have not switched to using the VPI library in their PLI applications:

- Portability to all major simulators
- Learning to use the new VPI library
- Performance of PLI applications

With the release of VCS 6.1, these reasons are mostly old-guard thinking.

Portability. Portability is no longer a critical issue. All major simulators now support most of the VPI library. Any PLI application that runs on VCS 6.1 will also work with NC-Verilog, Verilog-XL and ModelSim. The converse is not true, however. Since VCS 6.1 has not implemented the full VPI standard, an application that runs on another simulator may not work with VCS 6.1. This limitation will disappear quickly, however, as Synopsys adds support for more of the VPI library during 2002. It should be noted that some lesser-known simulators do not support the VPI, including Polaris, FinSim, Silos and VeriloggerPro. If a PLI application must run on these simulators, then the VPI library should not be used.

Knowledge. There is a learning curve involved with using the VPI library. VPI routines work very differently than the older TF and ACC libraries. The VPI library requires a more disciplined C programming style. PLI application developers who know the TF and ACC libraries need to learn to think differently in order to create efficient VPI-based PLI applications. However, the VPI library is simpler, more consistent, and better documented, and is therefore easier to learn than the older libraries. The author has taught PLI training courses at many different companies. Consistently, most engineers with good programming skills have expressed a preference for using the VPI library over the old, bloated TF and ACC libraries..

Performance. Simulation run-time performance of a PLI application may be a reason not to use the VPI library. Like the ACC library, the VPI library has arbitrary access to the entire simulation data structure. Simulation compilers cannot predict what information will be accessed, and there-

fore cannot optimize out information that will not be accessed by PLI applications. VCS deals with the arbitrary access issue by allowing the user of a PLI application to specify limits to the application's access. For example, the user can specify that the PLI can only access information within a certain module, or only within a certain instance of that module. Using these access restrictions, VCS can still efficiently optimize the simulation data structure. VCS also provides a mechanism for the simulator to determine what access is required, and generate the set of access restrictions automatically (using the `+learn` and `+applylearn` invocation options). With a properly restricted PLI application, performance, while a consideration, is not really a reason not to avoid using the VPI library.

7.0 Do DirectC, SystemVerilog and/or SystemC make the PLI Obsolete?

The Verilog PLI is not the only way to gain access to C and C++ from a hardware model. There are extensions to the Verilog language that access to the C and/or C++ languages and libraries, as well as high-level hardware modeling languages that utilize C or C++. Some examples are: Synopsys DirectC, SystemVerilog, and SystemC. These and other alternatives to the Verilog PLI raise the legitimate question as to whether using the VPI library is even necessary. To answer this question, the following paragraphs present a brief description of the nature and intent of some of these alternatives to the PLI.

DirectC. The Synopsys *DirectC*[6] is a set of extensions that allows C functionality to be directly called from Verilog source code, without having to go through the Verilog PLI. DirectC is proprietary to Synopsys, and is only supported by VCS. While DirectC can replace the PLI for many types of applications, it is not portable to other simulators. DirectC is ideal for applications that do not need to run on any other simulator.

SystemVerilog. *SystemVerilog*[7,8] is a proposed Accellera standard that extends the Verilog-2001 standard to include much of the C language, as well as many other features. SystemVerilog is a public standard, that is already being implemented by some simulator and synthesis vendors. The intent of SystemVerilog is to provide a new, higher level of modeling abstraction, which enables modeling much larger designs, and beginning the design process at an architectural level instead of an RTL level.

SystemC. In basic terms, *SystemC*[9] is an open-source library of C++ functions that allow modeling complex systems using the C++ programming language. With SystemC, the C++ models can be compiled and executed directly, without running a proprietary simulator. SystemC provides a high-level entry point for modeling large designs at an architectural level. As the design progresses towards implementation, however, the typical design flow is to convert SystemC models to Verilog or VHDL, in order to accurately represent and simulate hardware behavior.

The PLI is designed to allow two things: 1) to give the Verilog HDL user access to the C language, and 2) to give a C program access to the internals of the Verilog simulation data structure. The former allows Verilog test benches and models to access things such as the C file I/O library, the C math library, C inter-process communication pipes and sockets. The latter allows a PLI application the ability to analyze a simulation, do circuit tracing, dump value changes to a file, and perform other types of activity involving the internal simulation data structure.

DirectC and SystemVerilog, in a large part, provide the same capability as the first capability of the PLI, and do so much more elegantly than the PLI. These extensions to Verilog make it possible to mix the C language with the Verilog language without the need of creating system tasks and defining a complete PLI application. Many applications that are done with the PLI today could be better done if Verilog contained more of the C language. For example, an abstract Bus Functional Model written in C requires a PLI application in order for a Verilog simulation to communicate with the C model. Embedded software in a System On Chip design requires a PLI application to communicate between the hardware model and the software program that is executing. In these types of applications the PLI is both awkward to use, and is often a bottleneck in overall simulation performance. DirectC and SystemVerilog are good alternatives to the PLI in these cases.

Adding C to Verilog does not provide the second capability of the PLI—access to the internal simulation data structure. There are many other types of PLI applications where using the PLI is still the best approach, if not the only approach. A few lines of Verilog source code can create a very complex design hierarchy, with any number of levels of hierarchy, any number of instances of the same module, and any number of instances of different modules. Parameter redefinition can reconfigure and change the vector sizes, array sizes and many other properties of each instance of a module. All of this design complexity is easily accessed with the PLI. The same information would be difficult, if not impossible, to obtain directly in Verilog source code, no matter how much the Verilog language is extended. In addition, PLI applications can be compiled independently of any Verilog source code, and only object files distributed to end users. This affords a level of protection for intellectual property and sensitive information that adding C directly to the Verilog language would not provide.

8.0 Conclusion

There are many compelling reasons to use the VPI library (“PLI 2.0”) in future PLI applications. Some of the key reasons presented in this paper are:

- The VPI library is now well supported by all major simulators
- The VPI library is easier to learn than the older PLI standards
- The VPI library requires a more disciplined structured C programming style, which leads to PLI applications that execute more efficiently
- The VPI standard is better defined, which helps eliminate the differences in behavior in different simulators common with the older PLI standards
- Perhaps most importantly, the VPI library is the only way to access the many new features in the Verilog-2001 HDL, as well as new features in any future generations of the Verilog standard

The common concerns for not using the VPI library—portability, knowledge, and performance—are minor when compared to the many benefits gained by the VPI library.

Yes, you should seriously consider developing most future PLI application using the VPI library! *Existing PLI applications than need to work with the new features in the Verilog-2001 standard should also be converted to use the VPI library.*

9.0 Sources:

[1] *"IEEE Std. 1364-1995 standard for the Verilog Hardware Description Language"*, IEEE, Pascataway, New Jersey, copyright 1995. ISBN 0-7381-3065-6.

[2] *"IEEE Std. 1364-2001 standard for the Verilog Hardware Description Language"*, IEEE, Pascataway, New Jersey, copyright 2001. ISBN 0-7381-2827-9.

[3] *"Getting the Most out of IEEE1364-2000 Verilog Standard"*, Don Mills and Stuart Sutherland SNUG San Jose 2001 Conference, paper MC2_2.

[4] *"ReleaseNotes"*, from the VCS 6.1 beta doc directory.

[5] *"The Verilog PLI Handbook, Second Edition"*, Stuart Sutherland, Kluwer Academic Publishers, Boston, Massachusetts, copyright 2002. ISBN 0-7923-7658-7.

[6] *"DirectC: An interface between Verilog HDL and C/C++ in VCS"*, HDLCon 2002 paper 4.3, International HDL Conference, San Jose, California, March, 2002. Proceedings published by Accellera, Napa, CA.

[7] *"SystemVerilog 3.0: Accellera's extensions to Verilog, draft 4"*, Accellera, Napa, California, copyright 2002.

[8] *"Verilog, the Next Generation: SystemVerilog"*, HDLCon 2002 paper 4.4, International HDL Conference, San Jose, California, March, 2002. Proceedings published by Accellera, Napa, CA.

[9] *"What is the Open SystemC™ Initiative?"*, SystemC web site, January 2002, www.systemc.org.

10.0 Additional Resources:

[a] *"The Verilog PLI Handbook, Second Edition"*, Stuart Sutherland, Kluwer Academic Publishers, Boston, Massachusetts, copyright 2002. ISBN 0-7923-7658-7. A comprehensive user's guide and reference on the PLI, updated to reflect all the new features and capabilities of the IEEE 1364-2001 standard.

[b] *"Verilog 2001: A Guide to the New Verilog Standard"*, Stuart Sutherland, Kluwer Academic Publishers, Boston, Massachusetts, copyright 2001. ISBN 0-7923-7568-8. Provides an overview of 45 of the most significant new features in the IEEE 1364-2001 standard, with examples of proper usage.

[c] The Verilog-2001 Web Page, www.verilog-2001.com.