# Using PLI to Implement a User Defined System Task for Use with SILOS-X and Harmony

## Introduction

Although the IEEE 1364 Verilog standard provides a number of standard system tasks and system functions, designers sometimes need to use customized system tasks and functions provided by a vender or create their own using PLI (Programming Language Interface) routines. The PLI routines provided by the IEEE 1364 Verilog standard include TF routines, ACC routines and VPI routines. This application note does not cover the detailed usage of those PLI routines but discusses the flow of making a PLI library to help designers to create and use user-defined system tasks and functions with SILOS-X and Harmony.

This example illustrates how a user-defined task is named, implemented as a C routine, compiled, linked into PLI library, and invoked in the Verilog code. Figure 1 summarizes the flow of creating and invoking a user-defined system task/ function in SILOS-X and Harmony.

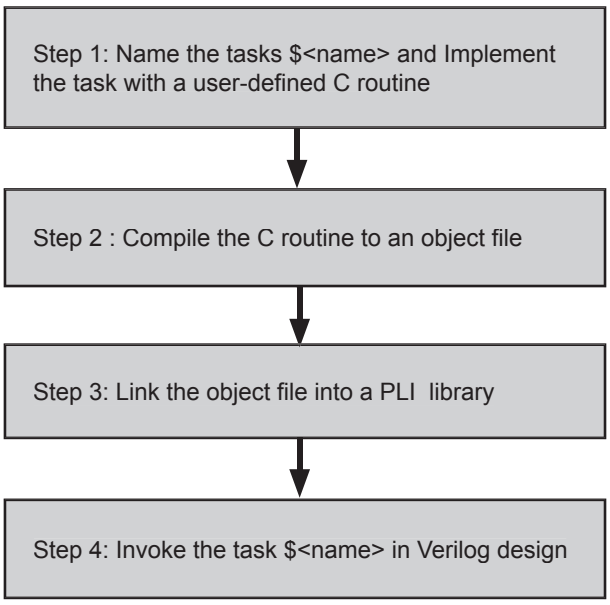


Figure 1. The flow of creating and invoking a PLI task on Silos-X and Harmony

## Example of how to create a PLI library and simulate it on the Linux platform:

**Step 1: Name and implement a user-defined C routine**
Depending on the design's purpose, designers can name their own user-defined tasks and implement them into C source code using PLI routines. SILOS-X and Harmony support TF routines and ACC routines to interface the user's code with the simulator. For information on those PLI routines, refer to the IEEE 1364 Verilog HDL specification that can be obtained from the IEEE.

User defined function C source code by using tf_putp() routine:

```
 /*////////////////////////////////////////////////
//
// title: C code for tf_putp() test to
write a single simple value
//
// This PLI C code uses tf_putp() to write
a simple value to a Verilog
// system task/function argument (param-
eters).
//
////////////////////////////////////////////////*/

#include <stdio.h>
#include "veriuser.h"

extern int putp_01_calltf();
char *veriuser_version_str = "";
int (*endofcompile_routines[])() =
{
    /*** my_eoc_routine, ***/
    0 /*** final entry must be 0 ***/
};

bool err_intercept(level,facility,code)
int level; char *facility; char *code;
{ return(true); }

stfcell veriusertfs[] = {
  {usertask,      0, 0, 0, putp_01_calltf,
```

```
0, "$putp_test", 1},
  {0}
};

/* Use tf_putp() with a single task/
function argument */
int putp_01_calltf()
{
  int val = 1;
  io_printf("PLI Code:   tf_putp(1, val) is
writing a value of %x (hex)\n",val);
  tf_putp(1, val);
  return(0);
}
```

In the C source code above, $putp_test is the name of the user-defined task which can be invoked in the Verilog code. putp_01_calltf() is the C routine to implement the tf_putp() routine ( PLI routine). $putp_test and putp_01_calltf() are linked at runtime through the veriusertfs table.

The veriusertfs table in the following C source code example contains entries in the s_tfcell structure.

```
stfcell veriusertfs[] = {
  {usertask,      0, 0, 0, putp_01_calltf,
0, "$putp_test", 1},
  {0}
};
```

A brief explanation of the fields in the veriusertfs table is in the veriuserstfs.h file:

```
/* VERILOG user tasks and functions C
header file */
typedef struct t_tfcell
{
    short type;          /* either usertask
or userfunction */
    short data;          /* parameter for
the following routines */
    int (*checktf)();    /* routine for
checking parameters */
    int (*sizetf)();     /* for providing
size of function return value */
    int (*calltf)();     /* routine called
during simulation */
    int (*misctf)();     /* miscellaneous
routine (see below) */
    char *tfname;        /* the name of the
system task or function */
    int forwref;        /* indicates spe-
cial parameters allowed */
    char *tfveritool;    /* Which Veritool
owns the task */
    char *tferrmessage; /* An optional
special case error message
                          which will be
printed if the task is skipped */

    /* these components are for system us-
age only */
    int hash;
    struct t_tfcell *left_p;
    struct t_tfcell *right_p;
    char *namecell_p;
    int warning_printed;  /* Flag is set
when skipping warning is printed */
} s_tfcell, *p_tfcell;
```

**Step 2: Compile the C source code to an object file ("*.o")**

In addition to the user's C source code file, there are three header files (acc_user.h, ext_user.h and veriuser.h) needed to compile this C source code. Contact Simucad ( support@simucad.com) to get these files.

Below is the command to compile the C source code to make an object file ("*.o").

```
gcc -fPIC -c myfile.c     # step to compile C source
```

The user may need to include other compilation flags depending on the situation. The source code must contain a single "veriusertfs" table that has entries for the user-defined functions in the source code.

**Step 3: Link to the PLI Library:**

The following command makes a PLI library named "mypli.so":

```
"ld -o mylib.so -dy -G myfile.o"     # step to load object
 file
```

**Step 4: Invoke the task $putp_test() in Verilog code and simulate**

After creating the .so file, use the "!pliload" command to specify the .so file for SILOS-X / Harmony. The "!pliload" command can be put in any Verilog input file, as long as it is not within a module boundary. Below is the example Verilog file using the user-defined task $ tf_putp():

```
////////////////////////////////////////////////
//
// title: testbench for tf_putp() test to
write a simple value
//
// This test writes a value to a scalar
reg data type
//
////////////////////////////////////////////////
`timescale 1ns/1ns
```

```
!pliload /full_path/mylib.so

module test;

  reg r1;

  initial
    begin
      #1
      $display("\nTest Bench: executing
\"$putp_test(r1);\" ");
      $display("Test Bench: expect tfarg 1
to receive 1 (hex)");
      $putp_test(r1);
      #1
      $display("Test Bench: tfarg 1 re-
ceived %h (hex)", r1);

      #1 $display("\n\n--------End of Test
Bench--------");
      $finish;
    end
endmodule
```

## Simulation Result:

The following is the simulation result running batch mode SILOS-X on a Linux machine.

```
[yunc@silos pli01_linux]$ gcc -fPIC -c
myfile.c
[yunc@silos pli01_linux]$ ld -o mylib.so -
dy -G myfile.o
[yunc@silos pli01_linux]$ silosx -b
Xlib:  extension "XFree86-Misc" missing on
display "shangrila:0.0".

       S I L O S - X  Version 4.10.5

 Copyright (c) Copyright Simucad Design Automa-
        tion 2007 All rights reserved.
  No part of this program may be reproduced,
  transmitted, transcribed,   or stored in a
 retrieval system, in any form or by any means
      without the prior written consent of

Simucad Design Automation, 4701 Patrick Henry Drive
     Santa Clara, California, 95054, U.S.A.
       (408)-567-1000  Fax: (408)-496-6080
         Web Site:   "www.simucad.com"
Ready: in pli01.v
Reading "/home/yunc/tmp/Silos/pli_example/
pli01_linux/pli01.v"
!pliload /home/yunc/tmp/Silos/pli_example/
pli01_linux/pli01.so
       Ready: sim
              Highest level modules (that
have been auto-instantiated):
                     test
              2 total devices.
              Linking ...

              1 nets total: 0 saved and 0
monitored.
              0 registers total: 0 saved.
              Done.

Test Bench: executing "$putp_test(r1);"
Test Bench: expect tfarg 1 to receive 1
(hex)
PLI Code:   tf_putp(1, val) is writing a
value of 1 (hex)
Test Bench: tfarg 1 received 1 (hex)


--------End of Test Bench--------
$finish in file "/home/yunc/tmp/Silos/pli_
example/pli01_linux/pli01.v" at line 26

              0 State changes on observ-
able nets.

              Simulation stopped at the
end of time 0.003us.
       Ready: exit
```

## Conclusion

The IEEE-1364 Verilog standard provides a wide range of PLI routines to extend the Verilog language so that designers can write their own system tasks and function for their design purpose. This application note describes the flow used to create a PLI library and invoke the user-defined task from standard Verilog code on SILOS-X and Harmony.

**Reference:**

[1] Verilog® HDL: A Guide to Digital Design and Synthesis, Second Edition.

[2] IEEE Standard Verilog® Hardware Description Language, IEEE Std 1364-2001.