Using the Verilog (PLI) Interface in Silos-X/Harmony on Windows

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 from vendors or create their own system tasks and functions by using the Programming Language Interface (PLI) routines. The PLI routines are provided by IEEE 1364 Verilog standard including 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 create and use user-defined system tasks and functions with Silos-X / Harmony.

We will discuss an example to illustrate how a user-defined task is named, implemented as a C routine, compiled and 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 on Silos-X/Harmony.

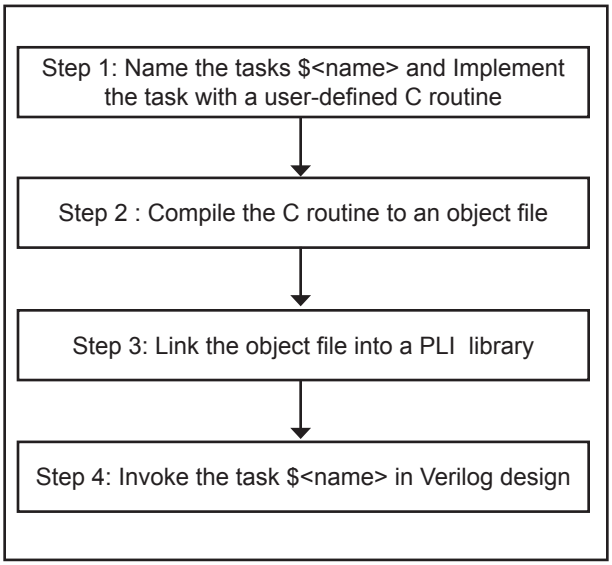

Figure 1. The flow of creating and invoking a PLI task on Silos-X / Harmony

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

Step 1: Name and implement a user-defined C routine

Depending on the design purpose, designers can name their own user-defined tasks and implement them into C source code using PLI routines. Silos-X / Harmony supports TF routines and ACC routines to interface the users code with the simulator. For information on those PLI routines, please refer to the IEEE 1364 Verilog HDL specification that can be obtained from the IEEE.

The following is a user defined function C source code example using the 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 (parameters).
//
////////////////////////////////////////////////////////*/


#ifdef _WIN32
#define DllExport __declspec(dllexport)
#else
#define DllExport
#endif

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

DllExport 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); }

DllExport s_tfcell veriusertfs[] = {
  {usertask,     0, 0, 0, putp_01_calltf,  0,
"$putp_test", 1},
  {0}
};


/* Use tf_putp() with a single task/function
argument */
DllExport 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 above C source code, $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 tf_putp() routine (PLI routine). $putp_test and putp_01_calltf() are linked at runtime through the veriusertfs table.

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

```
s_tfcell 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 sys-
tem task or function */
    int forwref;        /* indicates special pa-
rameters 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 usage
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;
```

Note: The C source code above defines the macro "DllExport" to add the exported functions and data to the output dll file. If this declaration is not used, the user must provide a .def file to define these exported names. Please refer to the Microsoft linker documentation for more information.

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

You will need to have a recent version of the Microsoft C/C++ compiler installed on your machine and your environment should be set up so that you can run the compiler and linker from a command window.

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. Please contact Silvaco (support@Silvaco.com) to get these files.

You will also need to copy the current version of the Silos-X dll import library ("libSilos-Xdll.lib") into your working directory. This file can be found in the Windows installation directory for Silos-X or Harmony, e.g. c:/sedatools/lib/Silos-X/4.10.39.R/x86-nt.

Open a command window and change directories to your working directory. Type the following command to compile the C source code and create an object file ("*.obj").

```
cl /ML /c myfile.c
```

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

Step 3: Link to the PLI Library:
Use the following command to make a PLI library named "mypli.dll":

```
link /dll libSilos-Xdll.lib myfile.obj
```

Step 4: Invoke the task $putp_test() in Verilog code and simulate
After you have created the .dll file, you will then need to use the "!pliload" command to specify the .dll 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

/*
 * "Silos-X_ms" is automatically defined by
Silos-X
 * on the Windows platform
 */
`ifdef Silos-X_ms
!pliload myfile.dll
`else
!pliload myfile.so
`endif

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 received %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 Windows machine.

```
E:\example\pli_appNote> cl /ML /c myfile.c

E:\example\pli_appNote> link /dll libSilos-
Xdll.lib myfile.obj

E:\example\pli_appNote>  c:\sedatools\exe\Si-
los-Xx –b pli01.v


E:\TestFiles\pli_appNote>l:\exe\Silos-X.exe -b
pli01.v

            Silos-X  Version 4.10.62.R

Copyright (c) 2004 - 2009 Silvaco Design Automa-
tion, All rights reserved.

Copyright (c) 1984 - 2004 Silvaco Data Systems,
All rights reserved.

                    4701 Patrick Henry Drive
              Santa Clara, California, 95054,
U.S.A.
                 (408)-567-1000  Fax: (408)-496-
6080
               Web Site:   "www.Silvaco.com"

Reading "pli01.v"
!pliload myfile.dll
```

```
Highest level modules (that have been auto-in-
stantiated):
                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 “E:\TestFiles\pli_appNote\pli01.v”
at line 30

        0 State changes on observable nets.

        Simulation stopped at the end of time
0.003us.
    No errors

E:\TestFiles\pli_appNote>
```

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 functions for their own design purposes. This application note describes the flow to create a PLI library and invoke the user-defined task from standard Verilog code on Silos-X / Harmony on the Windows platform.

Reference

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

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