

Integrating SystemC Models With Verilog Using The SystemVerilog Direct Programming Interface (DPI)

Stuart Sutherland

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

Objectives

- Introduce the SystemVerilog DPI
- Compare the DPI to the Verilog PLI
- Show more details on DPI capability
- Using the PLI to integrate Verilog and SystemC models
- Using the DPI to integrate Verilog and SystemC models
- Conclusions



- Each hardware design language has unique capabilities
 - Designs often comprise models from a mix of languages
 - This paper is not about what language is best
 - The paper focuses on mixing SystemC and Verilog models

This chart shows...

- SystemVerilog does not compete with SystemC
- SystemVerilog bridges a gap between Verilog and SystemC
- SystemVerilog enables easier usage of SystemC

- SystemC models can be integrated in Verilog simulations three different ways
 - Using proprietary methods
 - Most major simulator vendors provide a built-in mechanism to integrate SystemC models with their simulator
 - Advantage: Simple to use
 - Disadvantage: Not portable to other simulators
 - Using the Verilog Programming Language Interface (PLI)
 - Advantage: A standard that is portable to all simulators
 - Disadvantage: Difficult to do (more details later)
 - Using the SystemVerilog Direct Programming Interface (DPI)
 - Advantages: Simple to use and portable
 - Disadvantage: Restrictions on data types (more details later)

- The DPI is a new form of Verilog tasks and functions
- A Verilog task
 - Executes as a subroutine
 - Can advance simulation time
 - Does not have return values
 - Called as a programming statement
- A Verilog function
 - Executes as a function call
 - Must execute in zero time
 - Returns a value
 - Called as an expression

```

module chip (...);
  task sync_reset;
    resetN <= 0;
    repeat (2) @(posedge clock);
    resetN <= 1;
  endtask

  function int rotate (int a, b);
    return ({a,a} >> b);
  endfunction

  always @(negedge master_reset)
    sync_reset; //call task

  always @(a, b, opcode)
    case (opcode) //call function
      ROR: result = rotate(a,b);
      ...
    endcase
endmodule

```

- The SystemVerilog Direct Programming Interface:
 - “Imports” C functions into Verilog
 - Provides a new way to define a Verilog task or function

```

module chip (...);
  import "DPI" function real sin(real in); //sin function in C math lib
  always @(a, b, opcode)
    case (opcode) //call function
      SINE: result = sin(a);
      ...
    endcase
endmodule

```

Verilog code thinks it is calling a native Verilog task or function

- Using the SystemVerilog DPI
 - Verilog code can directly call C functions
 - Verilog code can directly pass values to and from C functions

What Comes Next

- ✓ Introduce the SystemVerilog DPI
- ❑ Compare the DPI to the Verilog PLI
- ❑ Show more details on DPI capability
- ❑ Using the PLI to integrate Verilog and SystemC models
- ❑ Using the DPI to integrate Verilog and SystemC models
- ❑ Conclusions



Verilog PLI Details



- Need to look at the PLI works in order to see if the DPI is better
- The PLI allows users to extend Verilog by creating “user-defined system tasks and system functions”

```
always @(a, b, opcode)
  case (opcode)
    SINE: result = $sine(a); // call a PLI application
```

- Must begin with a dollar sign (\$)
- Called the same as with Verilog tasks and functions
- The Verilog PLI is a *simulation interface*
 - Reads/modifies simulation data structures
 - Does not read Verilog source code
 - Does not work with synthesis compilers or other tools

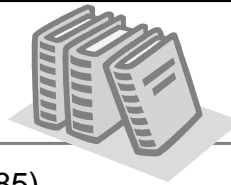
- A system task/function invokes a “calltf routine”
 - A user-defined C function associated with the task/function name
 - Can indirectly read system task/function argument values
 - Can indirectly return values back to simulation

```
always @(a, b, opcode)
case (opcode)
SINE: result = $sine(a);
```

With the DPI, this line of code can be called directly from Verilog!

```
int sine_calltf () {
double a, result;
a = tf_get(1); /* read $sine argument value */
result = sin(a); /* call C sin function */
tf_putrealp(0,result); /* write result to simulation */
return ();
}
```

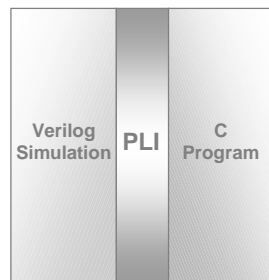
- Defining system task/function names and calltf routines is:
 - Complex
 - Done differently for each simulator



- TF library (introduced in 1985)
 - Provides access to system task/function arguments
 - Provides synchronization to simulation time and events
- ACC library (introduced in 1989)
 - Extends TF library to access to design structure
 - Created to enable ASIC delay and power calculation
- VPI library (introduced in 1995)
 - Superset of TF and ACC libraries
 - Adds access to behavioral and RTL models
 - Is the only library that supports new constructs in Verilog-2001 and SystemVerilog

The PLI Protects Simulators

- The PLI is a protecting layer between user programs and simulation data structure
 - Indirect access through PLI libraries
 - C program cannot directly read or modify the simulation data
 - Protects the simulator from bad C programs
 - Protects C programs from bad Verilog code



Halt! Who goes there?

Bad data is not allowed!

The Power of the PLI



- Supports system task/function arguments of any type
 - Data types, instance names, scopes, null arguments, etc.
 - Allows a variable number of task/function arguments
- Safe conversion of Verilog values to/from C values
 - Automatically converts any Verilog type to any C type
- Can find any modeling object anywhere in the data structure
- Can synchronize to simulation time
 - After blocking assignments, after nonblocking assignments, after all events, at future simulation times, etc.
- Can synchronize to any type of simulation event
 - Start, stop (breakpoints), finish, save, restart, logic value changes, strength level changes, RTL statement execution, etc.
- Supports multiple instances of system tasks/functions

The Disadvantages of the Verilog PLI

- Writing PLI applications is difficult to do
 - Must learn weird PLI terminology
 - Must learn what's in the PLI libraries
 - Must create checktf routines, calltf routines, etc.
- Linking PLI applications to simulators is hard
 - Multiple steps involved
 - Different for every simulator
 - Who does the linking...
 - The design engineer?
 - A CAE tool administrator?
 - Managing multiple PLI applications is difficult
- PLI code is seldom binary compatible
 - Must re-compile for every simulator



The DPI Makes Calling C Easy!



- The Direct Programming Interface makes it very simple for Verilog code to call C functions

```

module chip (...);
    import "DPI" function real sin(real in); //sin function in C math lib
    always @(a, b, opcode)
        case (opcode) //call function
            SINE: result = sin(a);
            ...
        endcase
endmodule
  
```

Verilog code directly calls the C function

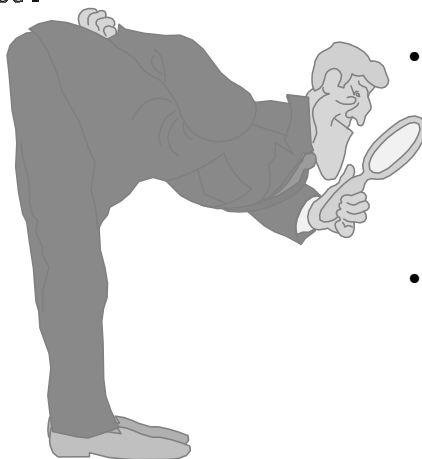
- Can directly call C functions from Verilog
 - Do not need to define complex PLI system tasks/functions
- Can directly pass values to and from C functions
 - Do not need the complex PLI libraries to read/write values

What Comes Next

- ✓ Introduce the SystemVerilog DPI
- ✓ Compare the DPI to the Verilog PLI
- ❑ **Show more details on DPI capability**
- ❑ Using the PLI to integrate Verilog and SystemC models
- ❑ Using the DPI to integrate Verilog and SystemC models
- ❑ Conclusions



A Closer Look at the SystemVerilog DPI



- The Direct Programming Interface originates from:
 - The Synopsys VCS DirectC interface
 - The Co-Design Cblend interface (now owned by Synopsys)
- The Accellera standards committee
 - Merged the features of each donation
 - Added more functionality
 - Ensured full compatibility with the IEEE 1364 Verilog standard
 - Added rules for binary compatibility between simulators

More on DPI Import Declarations

```
module chip (...);
    import "DPI" function real sin(real in); //sin function in C math lib
```

- Import declarations can be anywhere a function can be defined
 - Within a Verilog module
 - Within a SystemVerilog interface
 - Within a SystemVerilog package
 - Within a SystemVerilog “compilation unit”
- Import declarations must have a prototype of the arguments
 - Must exactly match the number of arguments in the C function
 - Must specify compatible data types (details on a later slide)
- The same C function can be imported in multiple locations
 - Each prototype must be exactly the same
 - A better method is to define one import in a package

Imported Function Arguments

- C functions can be imported as a Verilog task or function

```
import "DPI" function real sin(real in); //sin function in C math lib
import "DPI" task file_write(input string data, output reg status);
```

- The C function arguments can be imported as input, output or inout (bidirectional)
 - Arguments are assumed to be inputs unless declared otherwise

Task/Function Argument Data Types

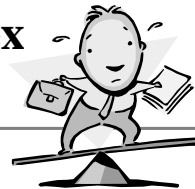
- The import declaration ***must*** specify SystemVerilog data types that are compatible with the actual C function data types

SystemVerilog types that map directly to C types

SystemVerilog types that require extra coding in C

SystemVerilog Data Type	C Data Type
byte	char
shortint	short int
int	int (32-bit)
longint	long long
real	double
shortreal	float
chandle	void*
string	char*
enum (using default int type)	int
bit (2-state type, any vector size)	abstract array of int types
logic (4-state type, any vector size)	abstract aval/bval pair arrays (like PLI)
packed array	abstract representation
unpacked array	abstract representation

Using More Complex Data Types



- Structures and unions
 - Can be passed to equivalent C structures and unions
 - Must use compatible types within the structure or union
- Arrays (unpacked)
 - Can be passed to equivalent C arrays
 - Must use compatible types within the structure or union
 - The array indexing might change (C must always starts with 0)
- Vectors of 2-state data types (packed arrays of bit type)
 - Must be converted to integer arrays in C
 - Requires lots of extra coding in C
- Verilog 4-state data types (packed arrays of reg, wire, etc.)
 - Uses the Verilog PLI aval/bval encoding
 - Requires lots of extra coding in C

Compatibility Warning!

Warning! Warning!



- It is the user's responsibility to correctly declare compatible data types in an import statement
 - Improper declarations can lead to unpredictable run-time behavior
 - The DPI does not check for type compatibility
 - The DPI does not provide a way to check and adjust for the data types on the Verilog side (the PLI can do this)

DPI Function Return Values

- A C function return value must be compatible with the basic SystemVerilog data types

Imported Return Type	C Function Return Type
byte	char
shortint	short int
int	int (32-bit)
longint	long long
real	double
shortreal	float
chandle	void*

Imported functions cannot return 1-bit values, vectors, structures, unions, or enumerated types

Warning!



- It is the user's responsibility to correctly declare an import statement that matches the C function return value
 - Incorrect import declarations can lead to unexpected behavior

Pure, Context and Generic C Functions

- A pure function result depends solely on the function inputs


```
import "DPI" pure function real sin(real, in);
```

 - Must have a return value; cannot have output or inout arguments
 - Cannot call any other functions or use static or global variables
 - Can be highly optimized for simulation performance **Advantage!**

- A context function is aware of the Verilog scope in which it is imported

```
import "DPI" context task print(input int file_id, input bit [127:0] data);
```

- Can be a void function, and can have output and inout arguments
- Can call functions from C libraries (for file I/O, etc.)
- Can call many of the functions in the PLI libraries (more on this later)
- A generic function is one that is not declared as pure or context
 - Can be a void function, and can have output and inout arguments
 - Can call other C functions
 - Cannot call most functions in the PLI libraries

Declaration Warning!

You must do this right!



- It is the user's responsibility to correctly declare pure and context functions
 - The DPI does not check for proper declarations
 - Improper declarations can lead to unpredictable simulation behavior
 - Improper declarations can lead to software crashes

Exporting Verilog Tasks and Functions

- Verilog tasks and function can be exported to C
 - Exporting allows C code to call Verilog code
 - C thinks it is calling a native C function
 - C functions can synchronize to Verilog time by calling a Verilog task with time controls

```
module chip (...);
  task sync_reset(inout resetN);
    resetN <= 0;
    repeat (2) @(posedge clock);
    resetN <= 1;
  endtask

  export "DPI" sync_reset;
  ...
endmodule
```

```
function void C_model()
{
  ...
  sync_reset(rst); // call Verilog task
  ...
}
```

Export declarations do not have argument prototypes

- C calling Verilog tasks and functions is unique to the DPI!
 - There is no equivalent in the PLI

Using the DPI to Simplify the PLI



- DPI capabilities can be extended by using the PLI libraries
 - A DPI context function can call many of the PLI library functions
 - Eliminates creating a system task/function name (e.g. \$sine)
 - Eliminates the complex PLI binding mechanism
- NOTE: The DPI context is not the same as the PLI context
 - DPI context is the scope in which the import declaration occurs
 - Matches the behavior of native Verilog task and functions
 - PLI context is the scope in which a system task is called
 - Context functions cannot fully utilize the PLI libraries
 - Cannot use PLI checktf and miscf routines
 - Cannot access all objects in the simulation data structure

Does the DPI Replace the PLI?

- Advantages of the SystemVerilog DPI
 - **Easy to use!**
 - *If* compatible data types are used on both sides
 - A direct interface between Verilog and C
 - Directly passes values to/from C using task/function arguments
 - Can be optimized for performance
 - *If* pure functions are used (limits what the function can do)
 - C can call Verilog tasks and functions
- Weaknesses of the DPI
 - Fixed number of arguments
 - User is responsible to pass compatible data types
 - Cannot access simulation data structure
 - Cannot synchronize to simulation events or event queue



Danger!



Does the DPI Replace the PLI?

- Advantages of the Verilog PLI
 - Allows access to entire simulation data structure
 - Find any Verilog object, anywhere in the design
 - Can synchronize to simulation
 - Synch to time (before or after blocking assignments, ...)
 - Synch to value changes, breakpoints, finish, ...
 - Provides an indirect access between Verilog and C
 - Protects simulation data structure from user C code
 - Automatically converts values between Verilog and C types
- Disadvantages of the Verilog PLI
 - Difficult to learn — even simple things are hard
 - Not binary compatible — different for every simulator
 - Can slow down simulator performance (significantly)



A Detailed Comparison of DPI versus PLI



- A paper presented at the SNUG-San Jose 2004 conference compares the DPI to the PLI
 - A table gives a side-by-side comparison of the major features of each interface

DPI Interface	TF Interface	ACC Interface	VPI Interface
Directly call C functions from Verilog code	Indirectly call C functions from Verilog code by associating the C function with a user-defined system task or system function name		
C function can call Verilog function or task	(no equivalent)		
(no equivalent)	Synch to simulator's event scheduler		Synch to simulator's event scheduler

The full paper is available at www.snug-universal.org/papers
(look for best technical paper at the San Jose 2004 SNUG conference)

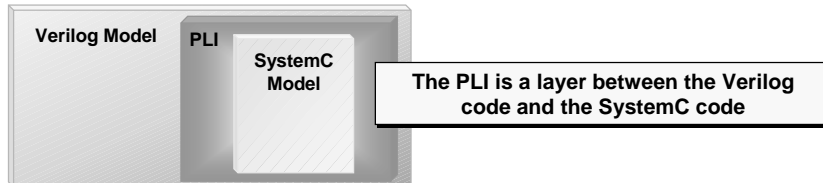
What Comes Next

- ✓ Introduce the SystemVerilog DPI
- ✓ Compare the DPI to the Verilog PLI
- ✓ Show more details on DPI capability
- Using the PLI to integrate Verilog and SystemC models**
- Using the DPI to integrate Verilog and SystemC models
- Conclusions



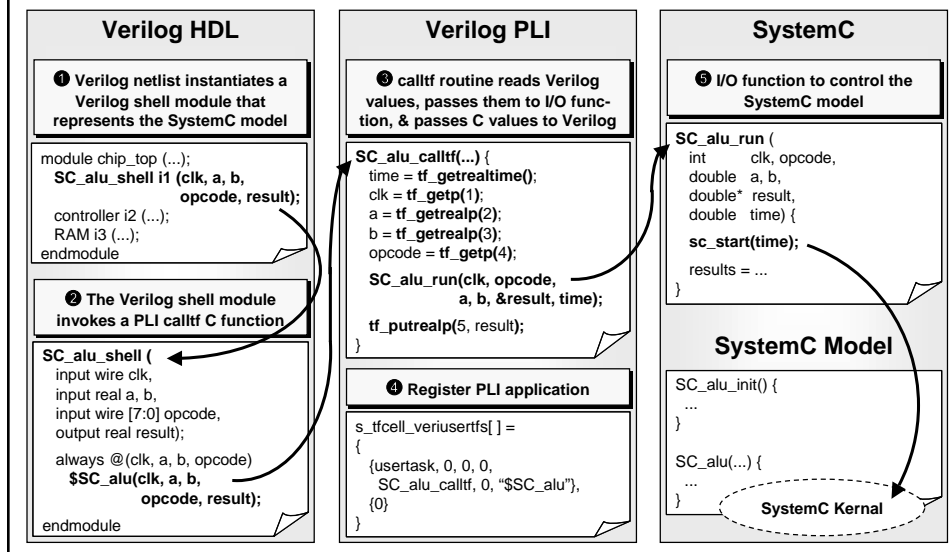
Using the PLI to Integrate Verilog and SystemC

- The Verilog PLI can be—and frequently is—used to interface SystemC models with Verilog models



- Verilog code must invoke a calltf C function that then connects to the SystemC model
 - Values must be *indirectly* passed to and from the SystemC model
- Using the PLI to interface Verilog with SystemC
 - Requires learning the PLI (at least the basics)
 - Adds the overhead of an intermediate layer (the calltf routine)

Verilog/SystemC Interface Using the PLI (partial code)



Pros and Cons of the PLI as a SystemC/Verilog Interface



- Pros
 - The Verilog PLI is a standard
 - Verilog to SystemC interfaces are portable to all simulators
 - Lots of user-experience available (examples, knowledge, etc.)
 - The PLI can safely and automatically convert values
- Cons
 - The PLI slows down simulation performance
 - The PLI is an extra layer between Verilog and SystemC
 - Extra C coding is required to write the PLI application
 - A knowledge of writing PLI applications is required

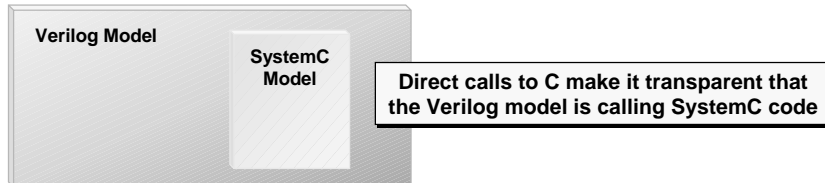
What Comes Next



- ✓ Introduce the SystemVerilog DPI
- ✓ Compare the DPI to the Verilog PLI
- ✓ Show more details on DPI capability
- ✓ Using the PLI to integrate Verilog and SystemC models
- Using the DPI to integrate Verilog and SystemC models**
- Conclusions

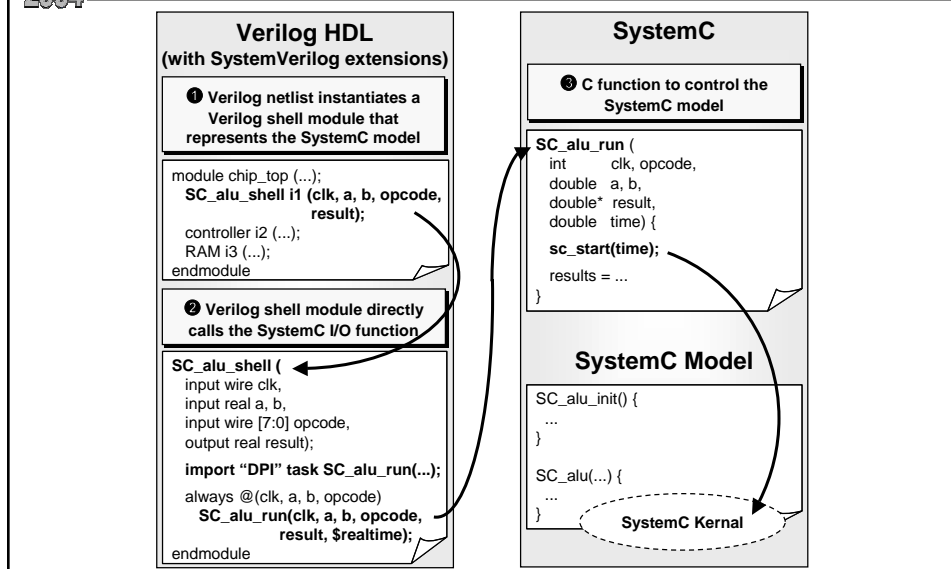
Using the DPI to Integrate Verilog and SystemC

- The DPI can be used to interface SystemC models with Verilog models



- Verilog code can directly call an imported C function that connects to the SystemC model
 - Values can be directly passed to and from the SystemC model
- Using the DPI to interface Verilog with SystemC
 - Eliminates the need for a complex PLI interface
 - Eliminates the need for proprietary, non-portable interfaces

Verilog/SystemC Interface Using the DPI (partial code)



Pros and Cons of the DPI as a Verilog/SystemC Interface

- Pros
 - The DPI is simple to use
 - Can directly invoke a C function to talk to the SystemC model
 - Can directly pass values between Verilog and C
 - The DPI requires no learning curve
 - All that is required is a simple “import” statement
- Cons
 - The DPI is only easy if compatible data types are used
 - `byte`, `shortint`, `int`, `longint`, `shortreal` and `real`
 - Extra coding required to manually covert values if `bit` or `logic` vectors, or other more complex data types are used
 - The DPI does not protect against improper import declarations
 - An incorrect prototype can crash the simulator

What Comes Next

- ✓ Introduce the SystemVerilog DPI
- ✓ Compare the DPI to the Verilog PLI
- ✓ Show more details on DPI capability
- ✓ Using the PLI to integrate Verilog and SystemC models
- ✓ Using the DPI to integrate Verilog and SystemC models
- **Conclusions**



Key Points

- SystemVerilog extends the Verilog HDL
- SystemVerilog compliments SystemC
 - The two languages have unique capabilities
 - SystemVerilog bridges the gap between SystemC and Verilog, making it easier to use SystemC in mixed HDL environments
- The SystemVerilog DPI simplifies the Verilog PLI
 - Verilog can directly call and pass values to C functions
 - C functions can call Verilog tasks and functions
- The DPI can interface SystemC models with Verilog simulations
 - Simple to do
 - A standard approach (making it portable to multiple simulators)
 - Requires using compatible data types to maintain simplicity

Questions?



Supplemental Slide: PLI Standards

- OVI (now Accellera) PLI 1.0 (1990)
 - Standardized the original TF and ACC libraries
- OVI PLI 2.0 (1993)
 - Intended to replace PLI 1.0
 - Not backward compatible, so never implemented in simulators
- IEEE 1364-1995
 - Standardized PLI 1.0 as TF and ACC libraries
 - Rewrite of PLI 2.0 to be backward compatible, called VPI library
- Accellera SystemVerilog 3.1 (2003)
 - Adds the Direct Programming Interface (DPI) to Verilog
- Accellera SystemVerilog 3.1a (projected for April 2004)
 - Extends VPI library to support all SystemVerilog constructs
- IEEE 1354-2005 (2006 ?) [projected date]
 - Will include SystemVerilog DPI along with many VPI extensions

Supplemental Slide: DPI versus PLI

- The Verilog PLI and the SystemVerilog DPI are both needed
- Use the DPI to
 - Directly call C functions that do not need to access the simulation data structure
 - Directly call PLI applications that only need limited access to the simulation data structure
 - Interface to C and SystemC models
- Use the PLI to
 - Access any object anywhere in the simulation data structure
 - Synchronize to the simulation event queue
 - Blocking assignments, nonblocking assignments, etc.
 - Synchronize to simulation events
 - Simulation start, stop, finish, save, restart, reset, etc.

Supplemental Slide: About the Author



- Involved with Verilog since 1988
- Considered a Verilog PLI expert
 - Wrote that 800 page book on the PLI
- Member of IEEE 1364 standards group since beginning
 - Co-chair of the Verilog PLI task force
 - Editor of the PLI sections of the standard
- Member of Accellera SystemVerilog standards group
 - Editor of the SystemVerilog standard
- Develops and presents expert-level Verilog, PLI and SystemVerilog training courses