



Leveraging System Verilog DPI for Co-Simulation

Allan Zacharda

Model Technology Incorporated



SystemVerilog C Interfaces



- **VPI Extensions Added for:**
 - **Assertions**
 - **Coverage**
 - **Datatrace API**

- **New Direct Programming Interface – DPI**
 - **Added specifically for Co-Simulating C/C++ and SystemVerilog**

Overview of DPI



- **DPI is a natural inter-language function call interface between SystemVerilog and C/C++**
 - Standard allows for other foreign languages in the future
 - DPI relies on C function call conventions and semantics
- **Golden Principle of DPI: On each side, the calls look and behave the same as native function calls for that language**
- **Binary or source code compatible across vendors**

Why DPI?

- **DPI provides a simple way of calling C functions from Verilog and getting results back**
- **DPI is bidirectional: C code can call Verilog tasks or functions and get results back**
- **VPI and PLI are not easy interfaces to use**
 - **Even trivial usage requires detailed knowledge**
 - **Many users do not need the sophisticated capabilities provided by VPI/PLI**

Some Example Uses of DPI



- **Value calculations done in C**
 - FFT, other numerical or crunching work
- **Complex I/O processing done in C**
 - Stimulus-fetching socket, custom file I/O, etc.
- **Test executives running in C**
 - Call export tasks to kick design into action and gather response after time has elapsed
- **Complex multi-language modeling**
 - Connect to SystemC or other multi-threaded environments running a portion of the verification

DPI Tasks and Functions



- Verilog tasks may...
 - consume time (wait, @, #, <= are allowed)
 - not be used in expression context
 - enable child tasks or call child functions
- Verilog functions may...
 - not consume time
 - be used in expression context
 - only call child functions, not tasks
- *DPI tf's have the same characteristics*
 - DPI tf's interact with Verilog tf's as if they were native

DPI - Declaration Syntax



■ Import tasks or functions (C functions called from SV):

```
import "DPI" [<dpi_import_property>][c_identifier=]  
            <dpi_tf_prototype>;
```

■ Export functions (SV functions called from C):

```
export "DPI" [c_identifier=] <dpi_tf_identifier>;
```

■ Explanation of terms

- <dpi_tf_prototype> same as a native task or function proto
- <dpi_tf_identifier> simple name of native task or function
- <dpi_import_property> -> *pure* or *context*
- c_identifier= is an optional C linkage name

Example Import Declarations



```
// The following defines a queue facility implemented in C code.
// SystemVerilog code makes use of it via import functions.
module queuePackage();

// Abstract data structure: queue
import "DPI" function chandle newQueue(input string queueName);

// The following import function uses the same C function for
// implementation as the prior example, but has a different SV
// name and provides a default value for the argument.
import "DPI" newQueue=
    function chandle newAnonQueue(input string s = null);

// Functions to round out the queue's interface
import "DPI" function chandle newElem(bit [15:0]);
import "DPI" function void enqueue(chandle queue, chandle elem);
import "DPI" function chandle dequeue(chandle queue);

// sequential code goes here . . .
...
endmodule
```

Example Export Declaration



```
interface ethPort( ... );
...

typedef struct {
    int unsigned packetType;
    int unsigned length;
    longint unsigned dest_addr;
    longint unsigned src_addr;
} etherHeaderT;

// The C code will name this export task "SendPacketHeader"
export "DPI" SendPacketHeader=handlePacketHeader;

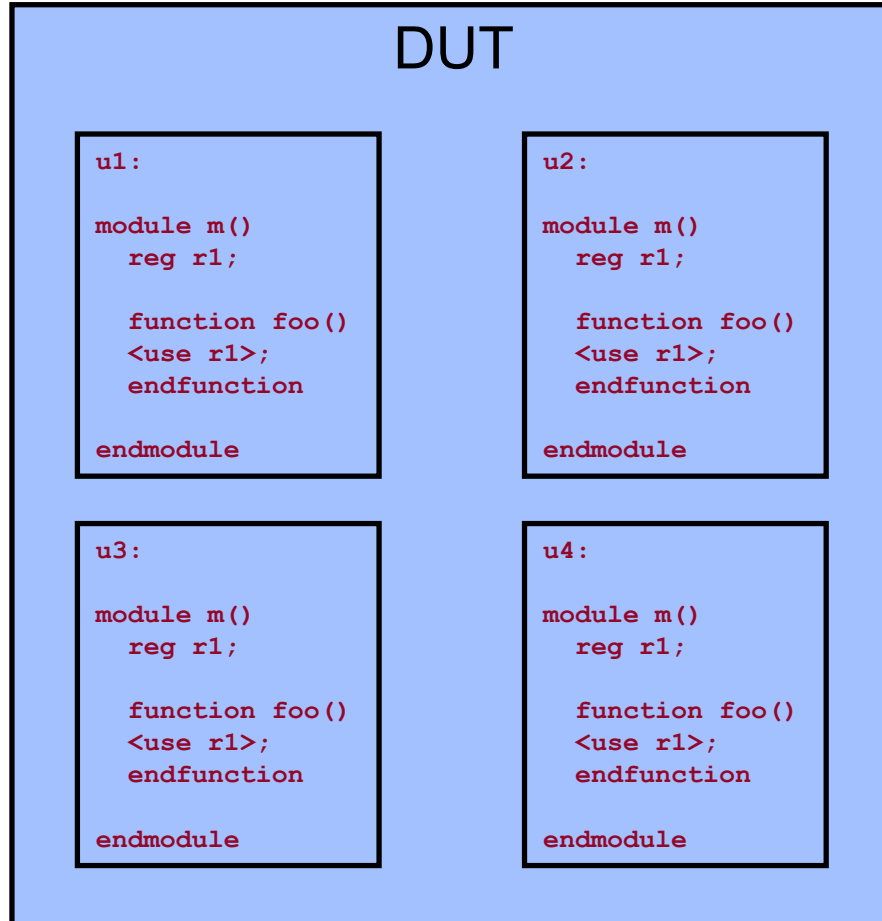
// Returns 32 bit CRC; callable from C code, may consume time
task handlePacketHeader(
    input etherHeaderT header);
    ...
    return computedCRC;
endfunction
...
endinterface
```

Import TF Properties



- The *pure* property:
 - result depends solely on inputs, optimizer might cache results
 - only applies to import functions, not tasks
 - is useful for compiler optimizations
 - has no side effects or state (I/O, global variables, PLI/VPI)
- The *context* property:
 - works with data specific to the enclosing module instance
 - useful when modeling system components
 - mandatory when PLI/VPI calls are used within the function
- Free functions (non-context): no relation to instance-specific data
 - Useful for doing calculations, i/o operations, numerical work, etc.
- Context import tf's are bound to a particular SV instance
- All export tf's are “context” tf's

What does “context” mean?



One definition of module m

+ One declaration of function foo

+ Four different instances of module m

= Function foo runs in four different *contexts*

Argument Passing in DPI



- Supports most SV data types
- Value passing requires matching type definitions
 - user's responsibility
 - packed types: arrays, structures, unions
 - unpacked aggregate types (arrays, structures, unions)
- Function result types are restricted to small values and packed bit arrays up to 32 bits
- Usage of packed types might prohibit binary compatibility

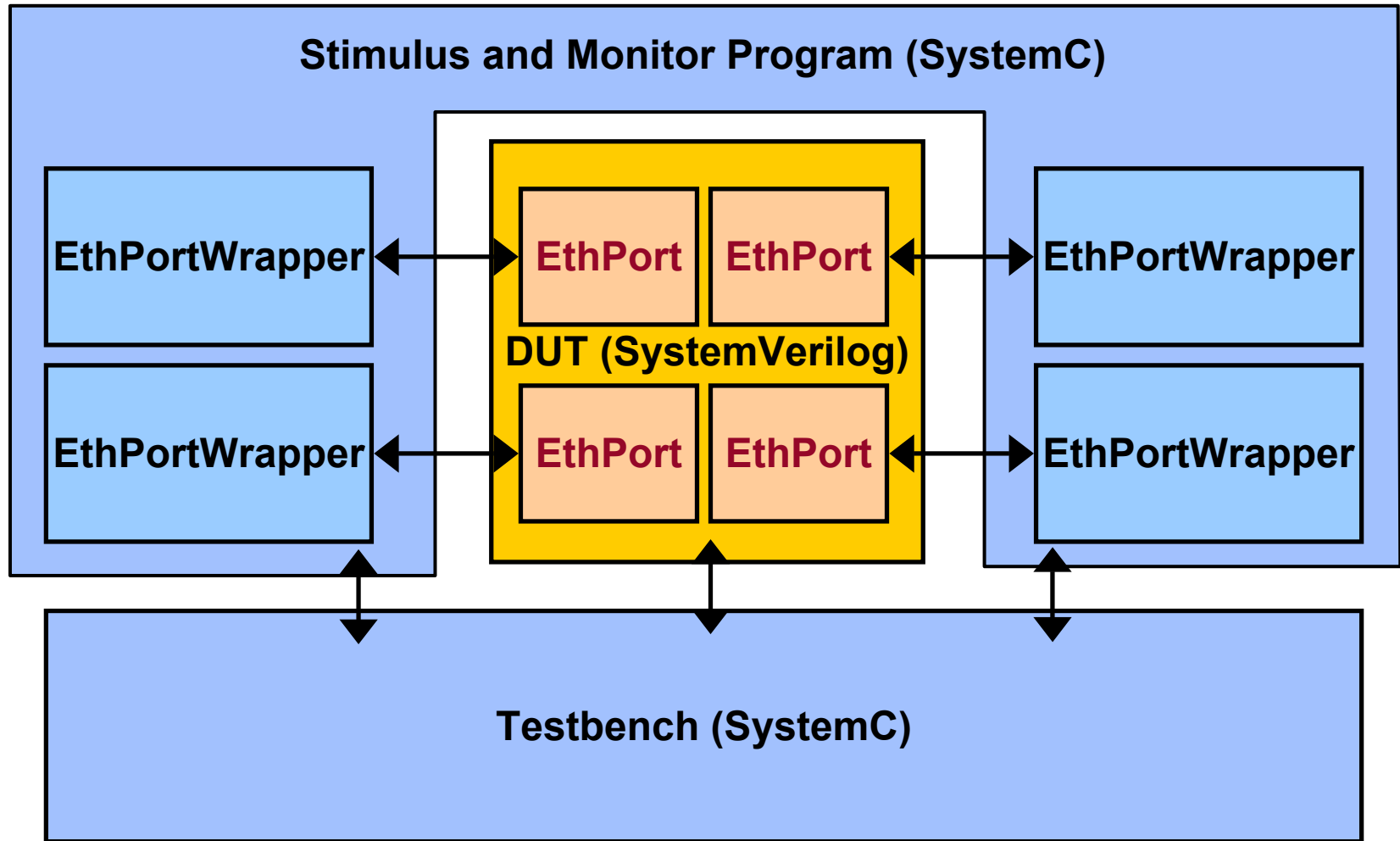
SV type	C type
char	char
byte	char
shortint	short int
int	Int (32-bit)
longint	long long
real	double
shortreal	float
chandle	void*
string	char*
bit	(abstract)
enum	int
logic	avalue/bvalue
packed array	(abstract)
unpacked array	(abstract)

Consistent Load of User C Code



- Only applies to DPI functions, not PLI/VPI
- All functions must be provided within a shared library
 - User is responsible for compilation and linking of this library
 - SV application is responsible for loading and integration of this library
- Libraries can be specified by switch or in a bootstrap file
 - `-sv_lib <filename w/o ext>`
 - `-sv_liblist <bootstrap>`

Ethernet Packet Router Example



C++: SystemC EthPortWrapper



```
1  #include "svc.h"
2
3  SC_MODULE(EthPortWrapper) {
4      private:
5          svScope myContext;
6          sc_module* myParent;
7      public:
8          SC_CTOR(EthPortWrapper) : svContext(0), myParent(0) { }
9          void Bind(const char* hdlPath, sc_module* parent);
10         void PutPacket(vec32* packet);
11
12         friend void HandleOutputPacket(svHandle context,
13             int portID, vec32* payload);
14     };
15
16     void EthPortWrapper::Bind(const char* svInstancePath, sc_module* parent) {
17         myParent = parent;
18         myContext = svGetScopeFromName(svInstancePath);
19         svPutUserData(myContext, (void*)&HandleOutputPacket, (void*)this);
20     }
21
22     void EthPortWrapper::PutPacket(vec32* packet) {
23         svSetScope(myContext);
24         PutPacket(packet); // Call SV function.
25     }
26
27     void HandleOutputPacket(int portID, vec32* payload) {
28         svScope myContext = svGetScope();
29
30         // Cast stored data into a C++ object pointer
31         EthPortWrapper* me = (EthPortWrapper*)svGetUserData(myContext,
32             (void*)&HandleOutputPacket);
33
34         // Let top level know another packet received.
35         me->myParent->BumpNumOutputs();
36
37         printf("Received output on port on port %\n", portID);
38         me->DumpPayload(payload);
```

SV-side: SV EthPort Module



```
1
2 module EthPort(
3     input [7:0] MiiOutData,
4     input MiiOutEnable,
5     input MiiOutError,
6     input clk, reset,
7     output bit [7:0] MiiInData,
8     output bit MiiInEnable,
9     output bit MiiInError);
10
11     import "DPI" context function void HandleOutputPacket(
12         input integer portID,
13         input bit [1439:0] payload);
14
15     export "DPI" void PutPacket;
16
17     bit inputPacketReceivedFlag;
18     bit [1499:0] inputPacketData;
19
20     //
21     // This export function is called by the C side
22     // to send packets into the simulation.
23     //
24     function void PutPacket(input bit [1499:0] packet)
25         inputPacketData = packet;
26         inputPacketReceivedFlag = 1;
27     endfunction
```

Summary



- **The SystemVerilog DPI provides a powerful and easy-to-use interface between languages**
- **DPI code is compatible across vendors**
- **Loading of DPI modules is done in a consistent manner**
- **DPI greatly simplifies integration of C/C++ design elements with HDL**