DPI Introduction
- Direct Programming Interface -

2020

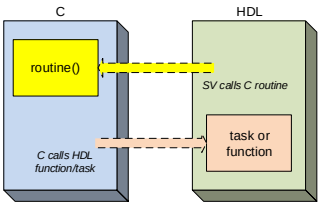
Ando Ki, Ph.D.
adki@future-ds.com

Table of contents

Language extensions of HDL

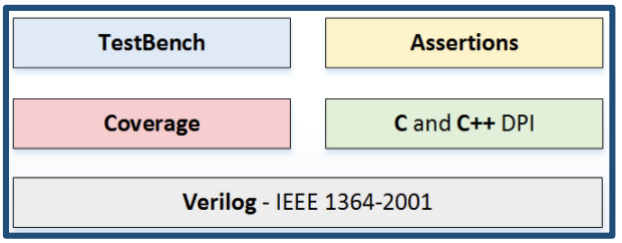
- HDL (Hardware Description Language) provides interfacing mechanism to high-level languages.

- Verilog
  - PLI (Programming Language Interface)
  - VPI (Verilog Procedural Interface)
- VHDL
  - FLI (Foreign Language Interface)
  - VHPI (VHDL Procedural Interface)
- SystemVerilog
  - DPI (Direct Programming Interface)

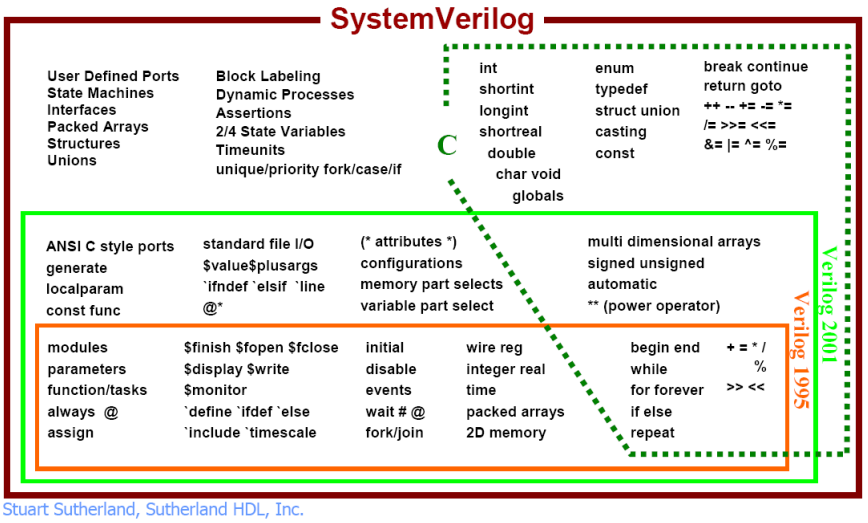


SystemVerilog

- SystemVerilog is a hardware description and ***Verification*** language (HDVL).
- SystemVerilog is an extensive set of enhancements to IEEE 1364 Verilog standards.
  - SystemVerilog is the superset of Verilog.
  - SystemVerilog has features inherited from Verilog HDL,VHDL,C, and C++.



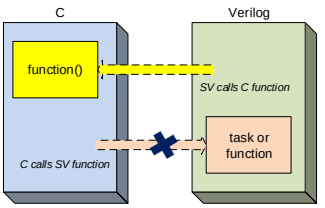
SystemVerilog and Verilog

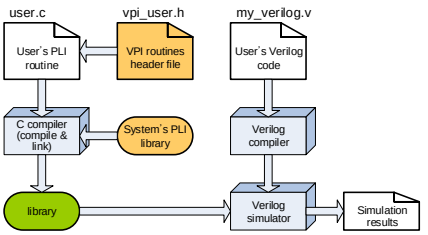


Stuart Sutherland, Sutherland HDL, Inc.

Verilog Procedural Interface (VPI)

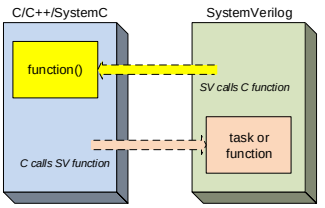
- Verilog PLI (Programming Language Interface) is a mechanism to invoke C or C++ functions from Verilog code, but not vice versa.
- VPI (Verilog Procedural Interface) routines are a super set of Verilog PLI (TF and ACC) routines.
  - Means of **calling the C model** in Verilog code.
    - *But the C model cannot invoke any Verilog codes.*
  - Means to **get the value** of the signals in Verilog code from inside the C code.
  - Means to **drive the value** on any signal inside the Verilog code from C code.





SystemVerilog Direct Programming Interface (DPI)

- SystemVerilog DPI (Direct Programming Interface) is an interface which can be used to interface SystemVerilog with foreign languages. These foreign languages can be C, C++, SystemC as well as others.
  - DPI is an inter-language function call interface between SystemVerilog and C/C++
    - The standard allows for other foreign languages.



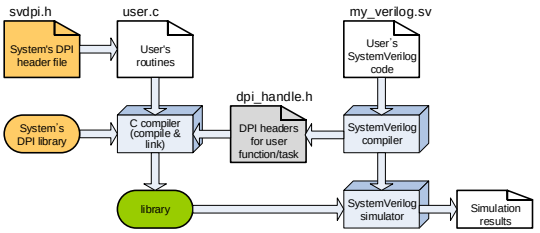
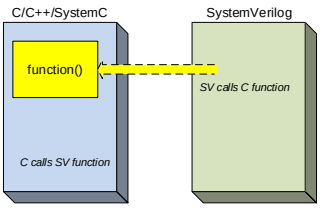
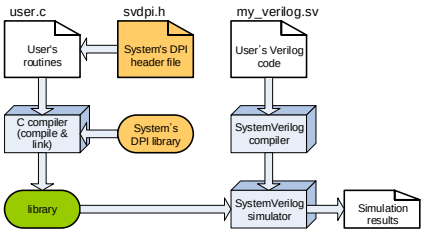


Table of contents





Syntax import method

import {“DPI” | “DPI-C”} [context | pure] [*sv_local_func_name* =]
[function | task] [*c_func_name*]([*port_list*]);

- import
- DPI or DPI-C
- context: may cause side effect by the C routine
- pure: returns value and cause no side effect
- sv_local_func_name
- function
- task
- c_func_name

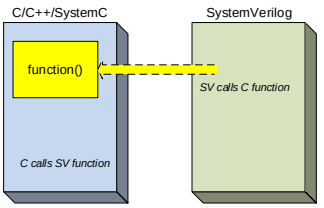
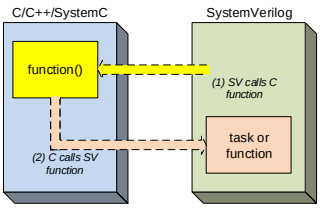
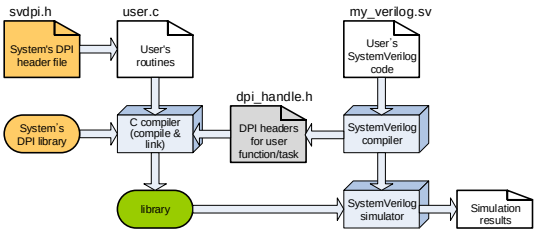


Table of contents





Syntax export method

export “DPI-C” [*c_identifier*=] [function | task] <*dpi_function_prototype*>;

- export
- DPI or DPI-C
- c_identifier
- function
- task
- dpi_function_prototype

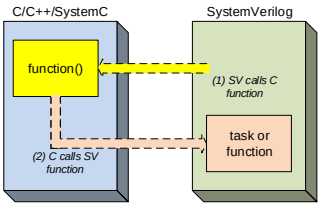


Table of contents

Overview of DPI

- DPI is a natural inter-language function call interface between SystemVerilog and C/C++
  - The standard allows for other foreign languages.
- DPI relies on C function call conventions and semantics
- On each side, the calls look and behave the same as native function calls for that language
  - On SV side, DPI calls look and behave like native SV functions
  - On C side, DPI calls look and behave like native C functions
- Binary or source code compatible
  - Binary compatible in absence of packed data types (svdpi.h)
  - Source code compatible otherwise (svdpi_src.h)

DPI - Declaration Syntax

- Import functions (C functions called from SV):
  - import “DPI“ <dpi_import_property> [c_identifier=] <dpi_function_prototype>;
- Export functions (SV functions called from C):
  - export “DPI” [c_identifier=] <dpi_function_prototype>;
- Explanation of terms
  - <dpi_function_prototype> same as a native function declaration
  - <dpi_import_property> -> pure or context (more later)
  - c_identifier= is an optional C linkage name
- Declarative Scopes of DPI functions
  - Import declarations -> same scope rules as native SV functions
  - Think of import functions as C proxies for native functions
  - Duplicate import declarations are not permitted, design-wide
    - Import declarations are not simple function prototypes
  - Export declarations -> same scope as function definition

Basics of DPI Arguments

- Formal arguments: input, inout, output + return value
  - input arguments shall use a **const** qualifier on the C side
  - output arguments are *uninitialized*
  - passed by value or reference, dependent on direction and type
- Shall contain no timing control; complete instantly and consume zero simulation time
- Changes to function arguments become effective when simulation control returns to SV side
- Memory ownership: Each side is responsible for its allocated memory
- Use of ref keyword in actual arguments is not allowed

Import Function Properties

- Possible properties of import functions are:
  - **pure**
    - useful for compiler optimizations
    - no side effects/internal state (I/O, global variables, PLI/VPI calls)
    - result depends solely on inputs, might be removed when optimizing
    - only non-void DPI function can be specified as pure.
  - **context**
    - mandatory when PLI/VPI calls are used within the function
    - mandatory when an import function in turn calls an export function
  - (default): no PLI/VPI calls, but might have side effects
- Free functions (pure, default): no relation to instance-specific data
- Context import functions are bound to a particular SV instance
  - Can work with data specific to that module / interface instance
  - One use of context import functions is to bind SV functions with complex object-oriented C verification systems (e.g. SystemC)
- Note that *all export functions are "context" functions*
  - Since they are in fact native SV functions defined in a specific declarative scope

Argument Passing in DPI

- Supports most SV data types
- Value passing requires matching type definitions
  - user's responsibility
  - packed types: arrays (defined), structures, unions
  - arrays (see next slide)
- 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
longint	long long
real	double
shortreal	float
handle	void*
string	char*
bit	(abstract)
enum	
logic	avalue/bvalue
packed array	(abstract)
unpacked array	(abstract)

Data type mapping correspondence

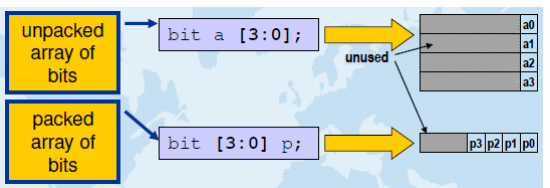
SV type to C equivalent		SV type to DPI-defined equivalent	
SystemVerilog	C	SystemVerilog	C
byte	char	bit	svBit
shortint	short int	bit[n:0]	svBitVecVal
int	int	logic	svLogic
longint	long int ; long long	reg	svLogic
real	double	logic[n:0]	svLogicVecVal*
string	char*	reg[n:0]	svLogicVecVal*
string[n]	char*	int[]	svOpenArrayHandle
chandle	void*	byte[]	svOpenArrayHandle
shortreal	float	shortint[]	svOpenArrayHandle
		longint[]	svOpenArrayHandle
		real[]	svOpenArrayHandle

Choosing DPI argument types

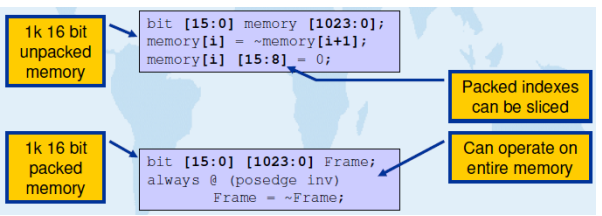
- Native C types, such as int and double, are good
- Composites (array, struct) of C types work well
- Use of the non-C types bit and logic:
  - Convenient for interfacing to legacy Verilog
  - More cumbersome programming needed
  - Binary and source compatibility issues
  - Worse for performance

DPI Array Arguments

- There are three types of array to consider
  - Packed array (elements of SV types “bit” or “logic”)
  - Unpacked array (elements of C-compatible types)
  - Open array (array bounds not statically known to C)



- Arrays use normalized ranges for the packed [n-1:0] and the unpacked part [0:n-1]
  - For example, if SV code defines an array as follows:
    - logic [2:3][1:3][2:0] b [1:10][31:0];
  - Then C code would see it as defined like this:
    - logic [17:0] b [0:9][0:31];



Open Array Arguments

- Open Array arguments have an unspecified range for at least one dimension
  - Good for generic programming, since C language doesn't have concept of parameterizable arguments
  - Denoted by using dynamic array syntax [] in the function declaration
  - Elements can be accessed in C using the same range indexing that is used for the SV actual argument
  - Query functions are provided to determine array info
  - Library functions are provided for accessing the array
- Examples:
- logic [] my1x3 [3:1];
- bit [] unsized_array [];

Argument Coercion for 'import' (1/2)

- import function arguments appear in 3 places:
  - Formal arguments in C-side function definition
  - Formal arguments in SV-side import function declaration
  - Actual arguments at SV-side call site
- Neither the C compiler nor the SV compiler perform any coercion at all between SV formals in an import function declaration and the corresponding formals in the C-side function definition
- Normal SV coercions are performed between SV actuals at a call site and SV formals in the import declaration
- User is responsible for creating C-side formal arguments that precisely match the SV-side formals in the import function declaration

Argument Coercion for ‘export’ (2/2)

- export function arguments appear in 3 places:
  - Formal arguments in SV-side function definition
  - Formal arguments in C-side function prototype
  - Actual arguments at C-side function call site
- Neither the C compiler nor the SV compiler will perform any argument coercion in the C-calls-SV direction
- The C compiler performs normal C coercions between C actuals at a call site and C formals in the function proto
- The programmer must provide C-side function prototype arguments that exactly match the type, width, and directionality requirements of the corresponding SV formals

C side library functions (1/2)

- A number of library functions are available that help you get information about the SystemVerilog side from the C side.
- **Library functions related to scope**
  - **Function:** *svGetScope*
    Use: Gets the current SystemVerilog scope of an imported function.
    Function prototype: *svScope svGetScope();*
  - **Function:** *svSetScope*
    Use: Sets the current SystemVerilog scope of an imported function.
    Function prototype: *svScope svSetScope(const svScope);*
  - **Function:** *svGetNameFromScope*
    Use: Gets the fully qualified current SystemVerilog path of an imported function from a scope handle.
    Function prototype: *const char* svGetNameFromScope(const svScope);*
  - **Function:** *svGetScopeFromName*
    Use: Sets the current SystemVerilog scope of an imported function.
    Function prototype: *svScope svGetScopeFromName(const char*);*

C side library functions (2/2)

- **Library functions related to packed arrays**
  - **Function:** *svGetSelectBit*
    Use: Reads a bit-select index i of an array reference svBitPackedArrRef of type Bit.
    Function prototype: *svBit svGetSelectBit(const svBitPackedArrRef s, int i);*
  - **Function:** *svPutSelectBit*
    Use: Writes the value of a bit s to a bit-select index i of an array reference svBitPackedArrRef of type Bit.
    Function prototype: *void svPetSelectBit(svBitPackedArrRef d, int i, svBit s);*
  - **Function:** *svGetSelectLogic*
    Use: Reads a bit-select index i of an array reference svLogicPackedArrRef of type Logic.
    Function prototype: *svLogic svGetSelectLogic(const svLogicPackedArrRef s, int i);*
  - **Function:** *svPutSelectLogic*
    Use: Writes the value of a bit s to a bit-select index i of an array reference svLogicPackedArrRef of type Logic.
    Function prototype: *void svPutSelectLogic(svLogicPackedArrRef d, int i, svLogic s);*
  - **Function:** *svSizeOfArray*
    Use: Returns the total size of an array in bytes or 0 if the array is not in C layout.
    Function prototype: *int svSizeOfArray(const svOpenArrayHandle);*
  - **Function:** *svGetArrElemPtr*
    Use: Returns a pointer to an element [index1][index2]... of an array or NULL if the array is not in C layout. The array does not have to be packed.
    Function prototype: *void *svGetArrElemPtr(const svOpenArrayHandle, int index1, int index2,...);*
  - **Function:** *svGetArrElemPtr*
    Use: Returns a pointer to an element [index1][index2]... of an array or NULL if the array is not in C layout.
    Function prototype: *void *svGetArrElemPtr(const svOpenArrayHandle, int index1, int index2,...);*

References

- 1800-2017 — IEEE Standard for SystemVerilog—Unified Hardware Design, Specification, and Verification Language.
  - 1800-2017, 1800-2012, 1800-2009, 1800-2005
- 1364-2005 — IEEE Standard for Verilog Hardware Description Language. 2006.
- 1364-2001 — IEEE Standard Verilog Hardware Description Language. 2001.
- 1364-1995 — IEEE Standard Hardware Description Language Based on the Verilog(R) Hardware Description Language. 1996.