COMPREHENSIVE SYSTEMVERILOG-SYSTEMC-VHDL MIXED-LANGUAGE DESIGN METHODOLOGY

Rudra Mukherjee
Mentor Graphics Corporation
rudra_mukherjee@mentor.com

Gaurav Kumar Verma
Mentor Graphics Corporation
gaurav-kumar_verma@mentor.com

Sachin Kakkar
Mentor Graphics Corporation
sachin_kakkar@mentor.com

ABSTRACT

Due to increased complexity in today's SoC designs, the importance of design reuse, verification, and debug becomes inescapable. SystemVerilog [1], VHDL [2], and SystemC [3] have unique strengths which make them more suitable to certain application domains. Mixed-language designs are proliferating because designers want use the powerful features of one language for creating test benches for designs written in the other language. Diversity of design teams with their different preferences, and integrating a growing number of IP blocks in a SoC, often written in different languages, also leads to a mixed-language scenario.

This paper provides guidelines and recommends an industry-proven, standardized way of writing a mixed-language design which will also aid in its integration in a mixed-language Verification Environment. The concept of SystemVerilog constrained testing and other SystemC verification offerings are combined with various VHDL constructs for design to build a verification environment with a common look and feel. This approach decouples the IP development from any stand alone RTL design development and ensures that the new IPs can operate and fully verified in mixed-language environment.

Beyond describing how to create and verify such IPs, this paper also describes how such compliant test benches are rapidly composed based on reusing existing components and other generic verification components, such as scoreboards and reference models.

The concepts presented in this paper provide a standard way of writing IPs and ensure that no glue is required. It allows components to be reused without modification in a scalable way. Restrictions on the use of certain features from certain languages provide uniformity and avoid some common pitfalls.

These features are comprehensive enough to provide an efficient framework for the design & verification of large and complex designs, all the way from the block to the system level.

1. INTRODUCTION

Current Design and Verification framework is very flexible and allows verification components to be written in many different ways, including different languages. As a result it can be difficult to combine and reuse existing verification components coming from different sources and it often requires a considerable amount of effort to glue them together.

While adding numerous new tools for verification and design to an engineer's toolbox, SystemVerilog, VHDL, and SystemC also challenges the engineer with a steep learning curve and an increased amount of complexity in the areas of languages, tools and methodologies; something that initially threatens to lower productivity, often significantly. To address these challenges, it is of great importance to be able to employ a high degree of reusability in designing and verification of components; which necessitates the need for a mixed-language environment.

Before starting the designing process, the unique strengths and weaknesses of different languages must be studies carefully to decide which language is most suitable for writing particular components of an IP. The first section of this paper discusses these unique strengths and weaknesses offered by the different languages.

The second section of the paper lays the foundation for supporting interactions between different languages by identifying equivalent types and defining rules for mapping these types. It discusses ways to import a package in any language irrespective of the language of origin. Importing a package across the language boundary makes all data-types, constants and functions available in the other language as if they were written in that language.

Section four presents all the different ways available today for making mixed- language connections, while discussing some common as well as not so common, but useful, techniques that designers can directly use to increase the efficiency of their mixed-language connections. Practical examples with scaled down versions of real world designs will be used to cover the aspects that designers need to understand while introducing mixed-language in their designs, and discuss methodologies to help him do so efficiently.

Section five discusses the guidelines and recommends an industry-proven, standardized way of writing mixed-language design and verification. The concept of SystemVerilog constrained testing and other SystemC verification offerings are combined with various VHDL constructs for design to build random verification environment with a common look and feel.

A customer case study is presented in section six of the paper highlighting productivity improvement results from real world mixed-language designs integration.

2. UNIQUENESS OF LANGUAGES

SystemVerilog [1], VHDL [2] and SystemC [3] have unique strengths which make them more suitable to certain application domains. This section discusses the unique strengths and weaknesses of different languages to help designers decide which language is most suitable for writing particular components of their IPs.

2.1 SystemVerilog

One of the most important languages to emerge for advanced design, verification, and modeling is IEEE STD 1800 SystemVerilog with its advanced verification, modeling and hardware design capabilities. It equips verification engineers with several tools like random constraints, assertions, and functional coverage to help write not only the designs but also test benches with ease.

SystemVerilog also offers an advanced support for interfaces, by offering it as a separate construct in the language. This, together with support for classes and inheritance facilitates multiple abstraction levels, which eases reuse of components.

It also offers named events, dynamic process creation, semaphores, mailboxes, associative arrays, and a direct C language interface, making it a popular language of choice for verification engineers.

2.2 VHDL

IEEE STD 1076-2002 VHDL needs no introduction. It has been, and still is in some parts of the globe, the preferred language of choice for writing designs. VHDL is a strongly and richly typed language. It is derived from the Ada programming language which makes it so much more verbose than SystemVerilog and VHDL, that the designs written in VHDL are often self-documenting. Since VHDL emphasizes on unambiguous semantics, race conditions, as a artifact of the language and tool implementation, are not a concern for VHDL designs.

The explicit support for physical types, FPGA library, ability to define signal/net resolutions and a strong support for configurations makes it a strong contender wherever these features are essential.

The new standards of VHDL are aimed at adding test bench and extended assertion capabilities to the language; the two areas where it lags behind SystemVerilog and SystemC.

2.3 SystemC

IEEE STD 1666 SystemC is another important language, with its powerful modeling features and tight links to the C/C++ programming languages. SystemC is essentially a set of C++ classes and macros which provide an event-driven simulation kernel in C++. Although it is strictly a C++ class library, SystemC is often viewed as being a language in its own right. Together with SystemC Verification Library (SCV), it offers several test bench features like random constraints, and assertions which make it the language of choice for writing test benches.

SystemC is generally used for system-level modeling, architectural exploration, performance modeling, software development, functional verification, and high-level synthesis. SystemC is often associated with Electronic system level (ESL) design, and with Transaction-level modeling (TLM).

3. LAYING THE FOUNDATION

The challenge faced by many SOC teams is how to use these languages together for mixed-language, mixed-abstraction level verification.

In this section, we will discuss using these three very capable languages for verification, RTL design, and modeling, in a mixed-language environment.

3.1 Sharing Types

Data types play a vital role in connecting mixed-language components. They form the backbone of signals that are used to transfer data across the language boundary. It is very important that the types connected across the boundary share an equivalent data representation model.

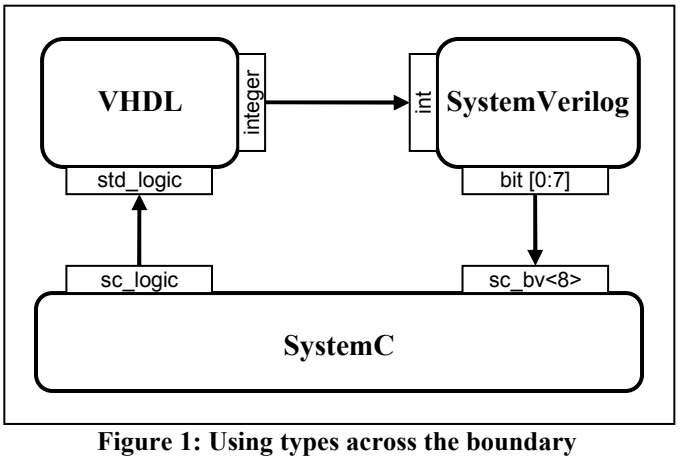

Figure 1: Using types across the boundary

While SystemVerilog and VHDL allow defining user-defined data types inside packages, SystemC allows defining user-defined data types inside header files. Designers should be able to directly include the equivalent object model for data types defined in packages (or header files), in any language to any other language, facilitated by the mixed-language design tools. For example, if an IP is written in VHDL, and uses types defined in a VHDL package, compatible data types can be imported to define SystemC variables and constants, while writing a test bench for that IP.

Example: SystemVerilog data types import

SystemVerilog Package

```
package sv_pack;
   typedef struct {
       int flags;
       bit[0:31] data;
       bit[0:31] addr;
   } record_t;
endpackage
```

SystemC Module

```
SC_IMPORT(std_logic_1164);
SC_IMPORT(vhdl_pack);

//Data-Types imported from SV
record_t in1;
const record_t a = {0, 0};
```

Table 1 explores compatible data types in SV, VHDL, and SystemC to assist designers and mixed design EDA tools to relate cross HDL data types.

Table 1: SV-SC-VHDL Compatible Data Types

SystemVerilog	VHDL	SystemC
bit logic reg wire	bit std_logic boolean	sc_bit sc_logic bool
bit vector logic vector reg vector wire vector	bit_vector std_logic_vector	sc_bv<n> sc_lv<n>
int	integer	int
shortint	integer	short
longint	integer	long long
byte	integer	char
enum	enum	enum
struct	record	struct
real	real	double
shortreal	real	float
char	character	char
class	N.A.	class
union	N.A.	union
shortint (W=16) int (W=32) longint (W=64) bit vector	integer (W=32) bit_vector	sc_int sc_uint sc_signed sc_unsigned sc_bigint sc_biguint

3.2 Operator Overloading

All three languages, i.e. SystemVerilog, VHDL and SystemC, allow operator overloading. The need for operator overloading arises from the fact that the operators in the language are defined only for operands of some predefined type. Operator overloading can be used in conjunction with data-type import to allow the defined arithmetic operators to be applied to the imported data types, such as structures, unions, classes, and enumerations.

Example:
SystemVerilog
```
bind == function ethernet_frame comp_frames (ethernet_frame,
ethernet_frame);
```

SC Module
```
SC_IMPORT(ethernet_ip_pack);

ethernet_frame frame1, frame2;
…
// implicit call to overloaded SV "==" operator
if (frame1 == frame2) ...
```

3.3 Procedures and Overloaded Functions

While VHDL provides subprogram facilities in the form of both procedures and functions, SystemVerilog provides these facilities in the form of tasks and function. SystemC extends all the functionality of C++ when it comes to support for functions. This allows reuse of procedures, tasks and functions in an efficient and intuitive manner.

While calling tasks, functions or procedures with complex argument types across the language boundary, equivalent data types can be specified as argument types to any of the argument types.

Example: SystemC using VHDL procedure.
VHDL Package
```
package vhdl_pack is
-- VHDL procedure declaration
procedure word_32_or (driver1, driver2 : in
word_32; en: in bit := '0'; or_out : out bit );
end vhdl_pack;
```

SC Module
```
SC_IMPORT(vhdl_pack);
...
void module_func() {
// function interface for VHDL subprogram call.
word_32_or(data1, data2, 1, &sum_or);
end
```

4. METHODS TO CONNECT MIXED-LANGUAGE IP BLOCKS

Often designers are not aware of various mixed-language design integration options. Other times the knowledge on various options is available, but it is difficult for a user to choose the best suitable option based on their mixed language design scenario. This difficulty in mixed-language IP integration and reuse often leads to finding issues late during the design cycle, which impacts the overall productivity. In this section we will introduce the five methods for making mixed-language connections and discuss their pros and cons.

4.1 Direct Instantiation

In the direct instantiation method, an IP block written in any language is instantiated directly inside the target IP block (written in any language) within the SoC. Here the instantiation statement follows the syntax of the target IP block, as if the instantiated IP block was written in the same language as the target IP block.

Example:
Clock Generator IP Block
```
module clockgen # (parameter period = 5 …)
(input period, input delay,  input duty_cycle, output clk);
  …
endmodule
```

Instantiating Clock Generator IP Block inside VHDL Stimulus Generator
```
clockgenInst: entity clkgenlib.clockgen port map(clk => clk);
```

4.2 Integration through Configuration

Configurations specify how a design is *assembled* during the elaboration phase of simulation. The sheer power offered by configurations offers incredible flexibility to designers while making mixed-language integrations.

A Verilog configuration can be referenced in the VHDL region as though it was a VHDL configuration, and vice versa. It can also be referenced in the configuration aspect of a VHDL component

configuration by just specifying a Verilog configuration name instead of a VHDL configuration name.

Example:
VHDL Entity top
```
entity top is
end entity;
architecture arch of top is
begin
  stiminst : vh_stimgen;
  clkinst : vh_clkgen;
end architecture arch;
```

Verilog Configuration vl_stimgen_config
```
config vl_stimgen_config;
  design work.vl_stimgen;
  instance vl_stimgen.gen_data use work.vh_gen_data;
endconfig
```

VHDL Configuration to Configure Instances stiminst & clkinst
```
configuration conf1 of top is
for arch
 for clkinst : vh_clkgen use entity work.vl_clkgen;
 end for;
 for inst1 : vh_stimgen use configuration work.vl_stimgen_config;
 end for;
end for;
end;
```

4.3 SystemVerilog Bind Construct

The SystemVerilog bind construct provides an IP block access to both external ports and internal signals in the target IP block. The selected and target IP blocks can be written in any design language. This method provides a powerful capability that, together with a specifically designed use-model, can be used to conveniently connect the two IP blocks independent of their languages. The SystemVerilog bind construct is increasingly becoming the preferred method for connecting IP blocks in SoC's today, as it offers hook-up connections between two IP blocks without requiring their source code to be present.

Example:
Clock Generator IP Block
```
module ClockGen # (parameter period = 5) (input output clk);
  …
endprogram
```

Binding Clock Generator IP Block to SystemC Stimulus Generator
```
bind StimGen ClockGen clockgenInst(.clk(clock));
```

4.4 SystemC Verification Connect

SystemC Verification Connect is a powerful construct that allows connection of signals across the hierarchy of a SystemC IP block to any other signal across the hierarchy of another IP block written in SystemVerilog or HDL. It can also be used on pre-compiled SystemVerilog and HDL IP blocks, but the SystemC IP block where scv_connect() constructs are used must have source-code visibility. This method cannot be used on compiled IP blocks.

Example:
Clock Generator IP Block
```
module clockgen #(parameter period = 5) (input output clk);
  …
endprogram
```

Using SCV_Connect to Connect Clock inside SystemC Stimulus Generator
```
scv_connect(clk, "/clkgeninst/clk", SCV_INPUT);
```

4.5 SC-DPI

The SystemC Direct Programming Interface (SC-DPI) method provides an interface between SystemVerilog and SystemC that facilitates inter-language function calls. This means a SystemVerilog IP can call a function defined in a SystemC IP, and vice versa. It is a fast and suitable technique of connecting SystemVerilog IP blocks with SystemC IP blocks that have their external interfaces defined in the form of methods only.

Example:
SystemC Stimulus Generator
```
sc_module (sc_stimgen) {
...
   return_status gen_stim(void);
...
};
```

Importing the SystemC Method 'gen_stim' in SystemVerilog:
```
import "SC-DPI" function return_status gen_stim();
```

5. RULES AND GUIDELINES

This section provides guidelines and recommends industry-proven, standardized ways of writing a design which will aid in its integration in a mixed-language Verification Environment, or with other blocks written in different languages in a SoC.

5.1 Choice of Hardware Verification Language

SystemVerilog with its advanced verification capabilities like random constraints, assertions, and functional coverage is recommended as language to write re-usable test benches for new IPs.

VHDL designs are recommended for designing IPs that is used in critical applications. Its unambiguous semantics reduces the chances of race conditions, glitches and timing violations in a design.

SystemC language is recommended wherever there is a need for tight linking with existing C/C++ based verification components. The authors do not recommend this language for synthesizable design because it based on C++ classes which are not yet synthesized by any industry standard tool. SystemC along with its various verification components (SCV) are recommended for Electronic system-level (ESL) modeling, architectural exploration, performance modeling, software development.

5.2 Inter-Language Communication

This section discusses the guidelines with the perspective of inter-language communication.

5.2.1 Type Compatibility

The most imperative rule for writing mixed-language ready designs is to always follow the mixed-language mapping table at the external interfaces, and avoid types that do not have any equivalent type in other languages. For example, sc_fixed types should be avoided at external interfaces of a SystemC design block, because they don't have any equivalence in SystemVerilog or VHDL. Connecting such blocks involve tricky wrappers which are more error prone.

Following the mapping table at the boundaries will allow components to be reused without modification from the block level to the system level in a scalable way. Restrictions on the use of certain features from certain languages provide uniformity and avoid some common pitfalls.

Authors also recommend re-use of common packages for sharing signal/variable values across mixed boundaries. Mixed-language designs should be able to directly include the equivalent object model for data types defined in packages (or header files). For example, if an IP is written in VHDL, and uses types defined in a VHDL package, compatible data types can be imported to define SC or SV variables and constants, while writing a test bench for that IP.

Section 3.1 of this paper should be referred for the recommended data types to facilitate re-use of the created design in a mixed-language environment.

5.2.2 Mixed language hierarchy

Authors recommend direct instantiation and SystemVerilog bind construct as the preferred ways of developing a hierarchy with mixed language components i.e. SV-VHDL, SV-SC, or SC-VHDL. There are other EDA vendor specific ways to allow communication between cross boundary components like system task, which are not considered here.

5.3 Language Semantics

Ensuring the correctness across mixed components is a key requirement and this can be ensured by avoiding the following at the external interfaces:

a) Events, classes, and other such types, which do not have any equivalence in all the languages, to appear at external interfaces.
b) Ensuring outputs at mixed boundaries are registered.
c) 2 state types being driven by 4 state types.
d) Multiple identifiers having the same case-insensitive name.
e) Reserved keywords of all other language.
f) Unnamed port declarations.
g) External interface defined as a SystemVerilog interface.
h) Parameterized types.
i) SystemVerilog parameters without types.

5.3.1 Classes

Classes should be avoided at external interfaces. However, classes offer some significant benefits because of which it sometimes becomes indispensable. If classes must be used at any external interface, designers must be aware that classes don't have any equivalent type in VHDL. As such, they'll have to do extra work while connecting the developed design in a mixed-language environment while connecting it to a VHDL block.

Classes allow coupling at a higher level of abstraction which enables complex components to work together seamlessly, obviating any extra piece of code required to map complex aggregate types to scalar data types at interfaces. It also helps designers to partition complex designs at any logical boundary that makes best use of both languages.

Since all class declarations must precisely correlate with classes declared across the language boundary, a class must not use constructs that are not available in classes in the other language. For example, SystemC classes with overloaded functions will not be allowed at the mixed-language boundary, as function overloading is not allowed in SystemVerilog.

Table 2 shows compatible class constructs in SV and SystemC to assist designers and mixed-language design EDA tools relate cross-HDL data types.

Table 2: Corresponding Class Constructs

SystemVerilog	SystemC
base class(extends)	base class(public)
virtual base class	virtual base class
class property	public data member
local class property	private data member
protected class property	protected data member
static class property	static data member
const class property	const data member
class method	public member functions
constructor(new)	constructor
static class method	static member function
parameterized classes	class template

Some SystemC class constructs (such as multiple inheritance, friend declarations, and copy constructors) do not have equivalent constructs in SystemVerilog and, hence, may not be allowed for classes crossing the language boundary.

5.3.2 Events

The equivalent of a SystemVerilog event is sc_event in SystemC, but there is no equivalent type for an event in VHDL. As such, use of an event should be avoided at external interfaces while writing a design. However, since an event equips designers with some very powerful features, which are not available otherwise, it cannot be completely dispensed with in some situations. An event should only be used at an external interface if it is guaranteed that the design will never be connected to VHDL in the VE or SoC.

An event passed across the SystemC-SystemVerilog interface follows the same semantics as used for passing regular data types. Passing an event leads to creation of an event in the other language. These two events point to the same underlying synchronization object, so triggering one will trigger its counterpart. As described in the SV LRM, assigning an SV event to the special value of NULL or to another event will break the link with the SystemC event.

5.3.3 Non-32 bit integer types

Since the only integer type available in VHDL is a 32 bit 2-state signed integer, integer types of sizes other than 32-bits (e.g. SV shortint, SV longint, SC short, SC long long, etc.) should be avoided at external interfaces to avoid issues arising due to range of data at the mixed-language boundary.

6. MIXED-LANGUAGE IN THE INDUSTRY

This section presents the results of our customer survey highlighting the challenges they face while integrating mixed-language in their designs, and the general trends that are prevalent in the industry.

A common observation that we made from customers across all regions is that they do not integrate mixed-language in their design by choice. Most of the time, they are forced to work on mixed-language boundaries in their designs either because they're

integrating a third party IP, or because they want to use some legacy code written in another language. There is a fairly good correlation between re-using components in a design and introducing mixed-language boundaries. Increasing the amount of re-use generally leads to an increase in the number of mixed-language boundaries in a design.

The biggest problem that designers face during mixed-language integrations is the lack of a standard. There is no LRM for mixed-language interactions, which forces designers to be dictated by the use models followed by EDA tools. These use-models vary from vendor to vendor, making their designs customized to a particular vendor. What works in one vendor's tool is not guaranteed to work exactly the same way in the other. What the industry desperately needs at this point of time is a standard way of making mixed-language connections.

Each language has its own set of rules and semantics, and the moment flow crosses the language boundary, rules of the other language take over. While a designer writes his blocks keeping in mind an environment written in a single language, and expects them to work in a certain way; the moment the other language takes over, they are not guaranteed to work in the same way. This leads to some intricate scenarios, which need careful handling on a case by case basis, often requiring some very interesting solutions like adding extra buffers at the boundary, creating extra wrappers, changing the method of connection, etc.

We found that the most popular methods of making mixed-language connections are direct-instantiation, because of the simplicity of its use-model, and SystemVerilog bind construct, because of the flexibility it offers to the users. Designers also often use VHDL configurations calling Verilog configurations, or vice-versa, in their designs to connect a Verilog region with a VHDL region.

Designers are also finding re-using packages across the SV-VHDL language boundary, i.e. either writing a package in SystemVerilog and including it directly in VHDL, or writing a package in VHDL and importing it directly in SystemVerilog, a very effective technique for mixed-language integrations.

7. CONCLUSION

The productivity of complex hardware system designs directly depend on reusable components. The reusability of cross language IPs is crucial in coping with the challenges in ASIC as well as FPGA domains. The proposed guidelines and recommendations on ways of writing a design will aid in its integration in a mixed-language Verification Environment, or with other blocks written in different languages in a SoC. The extensions to the type of reusability by importing cross-HDL packages (and SystemC header files) and raising the abstraction level during component reuse will result in complex mixed designs which will be far better and efficient than single HDL designs. The idea can be realized to take advantage of unique design and verification features offered by different languages without rewriting functionality available with legacy IPs.

8. REFERENCES

[1] SystemVerilog LRM IEEE 1800-2005 (www.systemverilog.com)
[2] VHDL LRM IEEE 1076-2002 (www.vhdl.org)
[3] SystemC LRM IEEE 1666-2005 (www.systemc.org)
[4] Rudra Mukherjee and Sachin Kakkar, "System Verilog – VHDL Mixed Design Reuse Methodology", DVCon 2006.
[5] Rich Edelmen, Mark Glassar, et al. "Inter Language Function Calls Between SystemC and SystemVerilog", DVCon 2007
[6] Rudra Mukherjee, Gaurav Kumar Verma, et al. "SystemC Mixed-HDL IP Reuse Methodology", IP-07
[7] Gaurav Kumar Verma and Rudra Mukherjee, "Adding New Dimensions to Verification IP Reuse", DVCon 2009