SystemVerilog Made Easy: A Perl Interface to a Full IEEE 1800 Compliant Parser

Debashis De and James Johnston
Verific Design Automation, Inc.

Since the adoption of hardware description languages (HDLs) as the methodology of choice for digital design in the early 1990s, an abundance of EDA tools based on the Verilog, VHDL and, later, SystemVerilog languages have been introduced. Providing full support of these languages for simulation and synthesis purposes turned out to be a large differentiator between EDA providers in the early days. But over time, as the industry started to better understand the languages and their implications, the quality of the EDA tools improved to a point where digital design engineers expect full support of the IEEE standards that define these HDLs [1] [2] [3].

While EDA companies invested many hours in supporting parsers, analyzers, and elaborators for VHDL and (System)Verilog, a different class of engineers working in semiconductor companies also made occasional attempts in writing parsers to solve a specific problem. Such in-house parsers tended to be single problem solutions that withered rapidly once the project is finished, lacking maintenance and full IEEE standard support.

Better solutions became available with the introduction of affordable off-the shelf parsers and elaborators, written and maintained by (System)Verilog and VHDL parser specialists. The availability of these packages speeded up development of HDL based products in EDA and FPGA companies significantly. Semiconductor companies with in-house tools also jumped straight in to support their internal flows.

Although this takes the burden of full IEEE language support off EDA developers, it still requires a fair knowledge of C++ to interface with the commercial parser and elaborator packages. For many hardware designers this turns out to be a bit of a stumbling block in

terms of productivity. Most of them are proficient in many types of scripting languages, be it csh, Tcl, Perl, or even Python, but have not had much experience with C++. If they encounter a project for which a language parser is required they much rather use their scripting skills than brush up on their C++.

To accommodate this, one could of course create a Perl or Tcl command for each C++ API inside the available parser / elaborator packages, but that sounds like a lot of work. Verific's software for instance contains well over 2,000 APIs. The good news is that the majority of these APIs can be developed with relatively little effort through the use of SWIG [4]. SWIG is an interface generator that connects programs written in C and C++ with scripting languages such as Perl. It works by taking the declarations found in C/C++ header files and using them to generate the wrapper code that scripting languages need to access the underlying C/C++ code. The Simplified Wrapper and Interface Generator (SWIG) is free software and the code that SWIG generates is compatible with both commercial and non-commercial projects.

At Verific, we undertook this exercise and created a Perl interface to our industry standard (System)Verilog and VHDL parsers / analyzers / elaborators (figure 1). Most of Verific's 2,000 plus application programming interfaces (APIs) translate rather easily using SWIG. But, we also found plenty of areas where we needed to step in. For instance, SWIG does not support Perl callbacks. Once we took care of the anomalies we ended up with a comprehensive Verific Perl package. It should be noted that this is a Perl API to a system defined in C++. All the underlying objects are written in C++ and there is a strong correlation between Perl and C++ APIs.

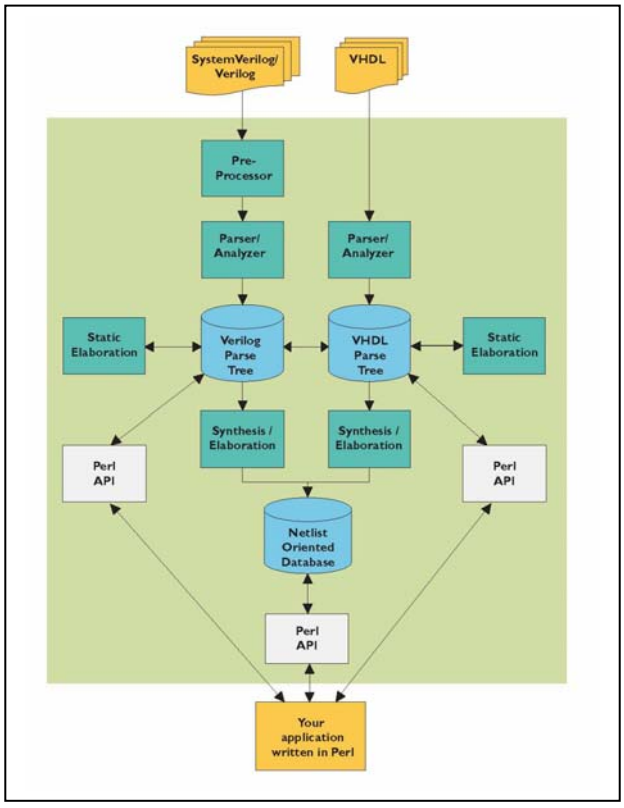

Fig. 1 Verific's standard parser technology extended with Perl APIs

A C++ API that looks like

```
type.method(params)
```

translates in Perl to

```
namespace::type_method(params)
```

and the objects returned from methods are pointers.

Special care needed to be taken of our container iterator macros, which we use widely in our C++ to iterate over modules, entities, cell, pins, you name it. In C++ one uses, for instance,

```
FOREACH_INSTANCE_OF_NETLIST(...)
```

whereas in Perl iterator wrappers needed to be provided that look like

```
$insts = $netlist->GetInsts();
while (my $inst = $insts->Next()) {
    ..do something with $inst
}
```

Once you're using the instance, the API is the same as in C++. A call to `Instance::method` simply becomes `$inst->method()`.

Figure 2 shows a rudimentary example of a C++ application and its equivalent in Perl. The application analyzes a SystemVerilog file ‘example.sv’, elaborates the design inside the file, and prints out the equivalent netlist.

```
#include "verific.h"

#ifdef VERIFIC_NAMESPACE
using namespace Verific ;
#endif

int main ()
{
    const char *file_name = “example.sv”;

   // Create a place-holder for the SV reader
   veri_file vrlg ;

   // analyze the SV file
   if (!vrlg.Analyze(file_name,"work", sv))
      return 1 ;

   // elaborate all analyzed design units
   if (!vrlg.ElaborateAll()) return 1 ;

   // get a handle to the top-level
   Netlist *top = Netlist::PresentDesign() ;

   // flatten down to primitives
   top->Flatten() ;

   // write the netlist
   VeriWrite veriWriter;
   veriWrite.WriteFile("output.v", top);
}
```

Sample C++

```
#!/usr/bin/perl
use Verific ;

$file_name = "example.sv";

# analyze the design
# in case of failure return
if (!Verific::veri_file::Analyze($file_name, 1)){
    exit(1);
}

# elaborate all analyzed design units
# in case of failure return
if (!Verific::veri_file::ElaborateAll()) {
    exit(1) ;
}

# get a handle to the top-level design
$top = Verific::Netlist::PresentDesign() ;

# flatten down to primitives
$top->Flatten() ;

# write the netlist
$writer = Verific::VeriWrite->new();
$writer->WriteFile("output.v",$top);
```

Sample Perl

Fig. 2 A side by side comparison of C++ and Perl for the same applications.

References

[1] IEEE Standard for SystemVerilog – Unified Hardware Design, Specification, and Verification Language
IEEE Std 1800™-2009

[2] IEEE Standard VHDL Language Reference Manual
IEEE Std 1076™-2008

[3] IEEE Standard for Verilog® Hardware Description Language
IEEE Std 1364™-2005

[4] www.swig.org

About the authors

Debashis De and James Johnston work in R&D at Verific Design Automation. They pride themselves on being part of the team that provides (System)Verilog and VHDL parsers / elaborators to a significant number of EDA developers and users worldwide.

Want to find out more ?

Verific will demo its SystemVerilog and VHDL Perl APIs in booth 2733 at the Design Automation Conference in San Diego, June 6 - 8. The Perl interface will be released to existing licensees in the June 2011 release.