Language Wars in the 21st Century: Verilog versus VHDL – Revisited

Steve Golson

Trilobyte Systems

Carlisle, Massachusetts

www.trilobyte.com

Leah Clark

San Diego, California

ABSTRACT

Back in the late 20th century, the VHDL versus Verilog debate was compared to a religious war that neither side could win. At various times, knowledgeable industry leaders have predicted that each HDL would prevail, but it seems we still live in a bilingual world. Is there still a language war?

In order to understand this conflict, we must first study where these languages came from and how they have evolved. Perhaps they aren't interchangeable, but instead can coexist, providing the right tool for the right job as needed. How does supporting more than one HDL affect your Synopsys flow? Has SystemVerilog changed the landscape?

Whether we continue to live in a dual-language world, or are approaching a time when one language will dominate, there are ramifications to be considered. We will discuss these concerns and much more, and hopefully we will answer the question once and for all of which HDL is "the winner".

Table of Contents

1. Introduction 4
2. A Brief History of Hardware Description Languages 4
   2.1 Early HDLs 4
   2.2 History of VHDL 5
   2.3 History of Verilog 5
   2.4 Language Wars in the Early 1990s 6
   2.5 Language Wars in the Late 1990s 7
3. HDLs in the 21st Century 9
   3.1 C for Hardware Design 9
   3.2 The Rise of Verification Languages 9
   3.3 Verilog and SystemVerilog 9
   3.4 VHDL 10
4. Languages in Today's Flows 10
   4.1 HDL versus RTL 10
   4.2 How Many HDLs Do We Have Today? Two, Three, More? 10
   4.3 Some Language Quirks 11
   4.4 Writing Mixed RTL 11
   4.5 Training 12
   4.6 Pros and Cons of Using Multiple HDLs for RTL 12
   4.7 Maintaining Mixed RTL 13
5. Implementation Challenges of Mixed Language Designs 13
   5.1 Reading in the RTL 14
   5.2 Synthesis and Physical Synthesis 14
   5.3 Logic Equivalence Checking 15
   5.4 DFT Tools 15
   5.5 Homegrown Tools 16
   5.6 And More 16
   5.7 You May Never Have Noticed Any of This… Yet 16
6. Verification Challenges 17
   6.1 Training – Revisited 17
   6.2 VHDL-2008 18
   6.3 Verification IP 18

7. EDA Infrastructure 18
 7.1 Coding Styles in General 18
 7.2 Support for New Features 19
 7.3 Case Study: Formality 19
 7.4 Case Study: Verific 20
8. Conclusions and Recommendations 20
 8.1 And the Winner Is 23
9. Acknowledgements 23
10. References 24

1. Introduction

Two hardware description languages—Verilog and VHDL—have become established as the industry-standard starting point for large-scale digital logic synthesis [1].

But why *two* languages? Wouldn't *one* be enough?

And most critically: *How does the need to support more than one language affect your design flow?*

The answer to this riddle is critically important to every design engineer, verification engineer, and engineering manager. Come with us as we search to find the answer! Along the way we will explore the history of HDLs, examine the intricacies of your current design flows, and discuss our recommendations for the future.

2. A Brief History of Hardware Description Languages

What is a *hardware description language* (HDL) and how is it used?

In 1977, G. J. Lipovski gave this simple definition: "…a hardware description language is a variation of a programming language tuned to the overall needs of describing hardware [2]."

In 1979, W. M. vanCleemput [3] listed the major applications of HDLs as:

- Description of the behavior and/or structure of a system as a means for accurately communicating design details between designers and end users.
- As the input to a system level simulator.
- As the input to an automatic hardware compiler.
- As the input to a formal verification system.

Amazingly, decades later these comments still hold true.

2.1 Early HDLs

In his 1940 master's thesis, Claude Shannon introduced a method for describing circuit behavior using Boolean algebra, manipulating these equations into their simplest form, and then synthesizing the corresponding relay switching circuits [4].

In a 1946 report, John von Neumann used a symbolic notation to describe the operations (i.e., instruction set) of the IAS computer [5].

Irving S. Reed in 1952 proved that Boolean algebraic equations can be physically realized as electronic circuits, and recommended "…in the initial synthesis [design] of a digital computer it is desirable to concentrate one's attention on the abstract model of the digital computer[1] [6]." By 1956, Reed extended this work into what he called a "register transfer language" to describe the design of a digital computer [7]. This is arguably the first true HDL, as it included the concepts of timing and clocks.

Research and development into hardware description languages continued through the 1960s. A 1974 survey listed over fifty HDLs from all over the world[2] [8].

[1] Or in modern terms, don't rush to gates!

[2] If you really must know, in the USA they were AHPL, APDL, APL, CASD, CASSANDRE, CDL, DDL, DSDL, FST, ISP, LALSD, LDT, LOGAL, LOTIS, MDL, PMS, RTL, RTS I, RTS II, SDL, SDL II, VDL. In Canada: CDL, SDL, DDL, NEDELA. In France: COSEQ, SISEQ, CASSANDRE. In West Germany: RTS I, RTS II, RTS III, ERES, AHPL. In Italy: CDL, ATLAS, DDS. In Japan: LDS, LORDS, DAIL 68, TBM, ADL. In Great Britain: REDAP 50, FLOG, CALL, DA70, SIMBOL, CDL, ASM, HILO, LOGOS, ARTHUR.

Confronted with such a proliferation of languages, in 1973 an international group of researchers attempted to consolidate existing HDLs into a standard language. This CONLAN project (named for CONsensus LANguage) was formalized into a working group in 1975. Their initial results were published in 1980 [9] followed by a more complete report in 1983 [10].

2.2 History of VHDL

The U.S. Department of Defense (DoD) began its ten-year VHSIC (Very High Speed Integrated Circuit) program in 1979 to address specific military needs in microelectronics [11]. As part of this project, it was thought that a broad-based standard HDL should be created to allow transfer of design data and descriptions independent of any given design tool or database, yet be machine-readable [12]. This new language was to be named VHDL (VHSIC Hardware Description Language).

Following a workshop held in the summer of 1981, the VHDL development contract began in 1983 and was awarded to a team from Intermetrics, IBM, and Texas Instruments. One major requirement was to use Ada constructs wherever possible [13]. VHDL was heavily influenced by CONLAN [14]. TI-HDL from Texas Instruments was used as a strawman language during the initial definition of VHDL [15].

The intention was to make VHDL a standard language, and mandate its use for design and description of DoD hardware [13]. This standardization process began in 1986 using VHDL version 7.2 as the baseline specification [16]. Several more iterations followed, culminating in IEEE Standard 1076-1987 [17].

In September 1988, MIL-STD-454L was issued by the U.S. Department of Defense [18]. Requirement 64 of MIL-STD-454L stated that all ASICs designed after September 1988 "shall be documented[3] by means of structural and behavioral VHDL descriptions." In 1992, VHDL became a U.S. government-wide standard when Federal Information Processing Standard 172 (FIPS Pub 172) was issued [19].

2.3 History of Verilog

Verilog was originally a proprietary verification/simulation product from Gateway Design Automation. Phil Moorby working with Chi-Lai Huang developed the language specification during the month of December 1983. Work on the logic simulator continued during 1984, and the first sale was in early 1985 [20].

Moorby's previous work with HILO-2 [21] was a major inspiration for the Verilog language, with influence also from C, Pascal, and especially occam [14]. Original goals for the language were to support logic simulation, fault simulation, and logic synthesis. The simulator supported min-max static and dynamic timing analysis [20].

By 1987, Moorby had written an even faster logic simulator called Verilog-XL, which boasted gate-level simulation speeds approaching that of hardware accelerators. Verilog-XL was a landmark product in EDA, rapidly gaining market share in an industry where there were several competing digital simulators. In addition to greater simulation speed, Verilog-XL boasted greater design-size capacity than its competitors. It featured a single, integrated language for modeling both the design and the testbench, plus the ability for end users to extend the language using Verilog's Programming Language Interface (PLI). Furthermore, Gateway actively and successfully courted ASIC vendors to use Verilog-XL as their timing signoff "golden simulator." At the request of these

[3] Note the requirement is for *documentation* not *design*. Many suppliers took advantage of this and created their design using other methods (sometimes even other HDLs), and then relied on automated translation tools to generate the required VHDL "documentation."

vendors, Gateway implemented several Verilog language enhancements to support accurate gate-level timing, including pin-to-pin path delays and a standardized method for back-annotation [22]. Due to Verilog being a proprietary language, Gateway could easily add the features that ASIC vendors wanted, and do it quickly without the multi-year process of IEEE standardization.

Although Verilog remained proprietary to Gateway, multiple companies licensed the language to use for ASIC cell simulation libraries as well as logic synthesis tools.

In late 1989, Gateway was acquired by Cadence, which continued to sell Verilog-XL and other Verilog tools obtained from Gateway [27].

2.4 Language Wars in the Early 1990s

By the late 1980s the stage was set for what became known as the "language wars," pitting Verilog versus VHDL.

VHDL became popular with many users and EDA tool vendors because it was an IEEE standard and thus could be used with no royalty cost. Also US military suppliers were required to document their designs using VHDL.

However, early VHDL tools revealed inconsistencies and ambiguities in the 1987 standard, and in 1991 the IEEE was forced to issue an unusual "Standards Interpretation" document [23]. Different vendors had dissimilar interpretations of the standard, and this became a source of frustration for early VHDL users [19].

In 1988 the 1st VHDL Users Group meeting was held as a "birds-of-a-feather" session at DAC (Design Automation Conference). In the fall of 1988, an independent VHDL Users Group meeting was held, and semiannual meetings continued afterwards. In 1991, the organization VHDL International (VI) was founded and the conference was renamed VIUF (VHDL International Users Forum) [24].

Meanwhile, Verilog continued to gain popularity with ASIC vendors who settled on Verilog-XL as their "golden simulator" for timing verification (i.e., signoff). Verilog-XL offered a standard timing back-annotation procedure, whereas VHDL specified no such standard, thus each VHDL simulator vendor used a different scheme [25].

Synopsys licensed the Verilog language from Gateway (later Cadence), and in 1988 introduced its Logic Compiler synthesis tool (soon renamed Design Compiler) [26]. By 1989, eight ASIC vendors supported Design Compiler. By the summer of 1991, 27 ASIC vendors supported the Synopsys synthesis tool, with 20 using Design Compiler at their own design centers [27]. Despite its early use of Verilog, Synopsys became a strong advocate for VHDL, perhaps because they offered a VHDL simulator product but no Verilog simulator.

In 1990, Cadence released the Verilog language to a newly formed nonprofit organization called Open Verilog International (OVI). The language definition entered the public domain and became available to any vendor [22]. Starting in 1992, OVI began sponsoring the annual International Verilog Conference (IVC). By 1992 there were about six U.S. companies who had announced or were developing Verilog simulators [14]. In 1993, Verilog had twice the market share of VHDL [28].

Nevertheless, by 1992, VHDL was enjoying widespread support. The 1993 IEEE VHDL standard officially clarified the language ambiguities and introduced a few new features [29]. The problem of inconsistent ASIC library models and different back-annotation methodologies across vendors was finally solved with the VITAL (VHDL Initiative Towards ASIC Libraries) IEEE standard issued in 1995 [30].

Joe Costello (CEO of Cadence) gave the keynote speech at the spring 1992 VIUF. He suggested that OVI and VI work together and called for an end to the "HDL wars." Furthermore he said that

Cadence was 100% committed to supporting both languages [27].

The conventional wisdom of the early 1990s was that VHDL was going to win the "language wars" [81]. Industry analysts such as Ron Collett and Gary Smith predicted VHDL revenues would overtake Verilog in 1992–1994 [31]. A 1992 survey by MJ Associates predicted Verilog usage would decline, and that VHDL would command 75% of the market by 1997 [32]. Even Phil Moorby, Verilog's inventor and the first Cadence Fellow, by 1992 was working on a VHDL simulator [14].

Consider Figure 1, which shows an n-gram plot of how often the words VHDL and Verilog appear in the Google Books corpus [33] for the years 1980–1995. Clearly many more VHDL than Verilog books are published in the 1980s and early 1990s.

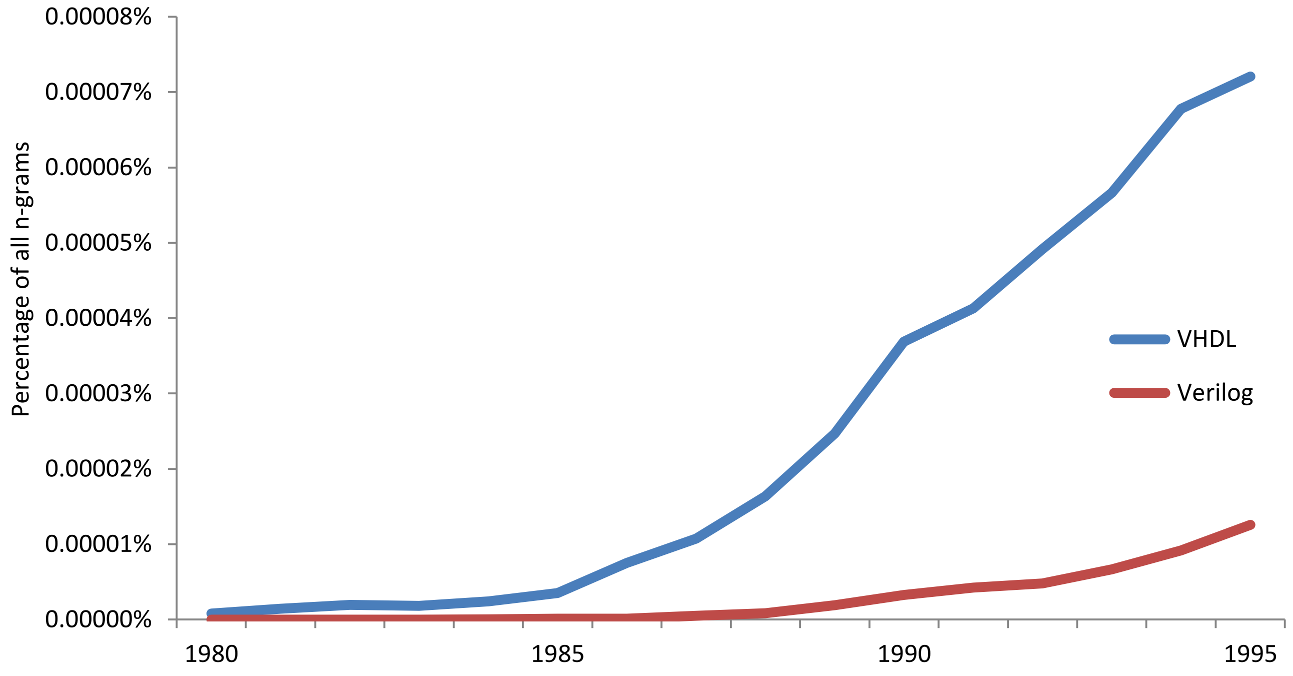

Figure 1: n-gram plot of VHDL and Verilog appearances in Google Books corpus

2.5 Language Wars in the Late 1990s

Despite the predictions, Verilog continued to prosper.

In 1992, OVI began working towards establishing Verilog as an IEEE standard. The working group held its first meeting in late 1993 [34].

By 1995, every major CAD vendor was supporting Verilog. Simulators, synthesizers, and tools were available from over 40 companies [28].

At the Synopsys Users Group meeting (SNUG) in early 1995, John Cooley hosted a 90-minute design contest. Using either Verilog or VHDL the contestants were to create a gate netlist for the fastest fully-synchronous loadable 9-bit increment-by-3 decrement-by-5 up/down counter that generated even parity, carry and borrow. This contest unexpectedly became fodder for the HDL language wars, as eight of the nine Verilog designers completed their netlists, while *none* of the five expert VHDL designers even got to gates [35]. This triggered enormous discussion about whether these results were indicative of the relative quality and usefulness of the two HDLs [36]. Was this evidence that Verilog really was better than VHDL?

Also in early 1995, DoD standard MIL-STD-454L was replaced and the use of VHDL was no longer mandated. Instead, the new wording stated that ASIC designs "should be documented" by means of VHDL [37]. By 1997 even this suggestion was removed [38].

In December 1995, Verilog became an IEEE standard [34].

The fall 1995 VHDL International User Forum (VIUF) had a difficult time attracting attendees [39]. In contrast, the International Verilog Conference (IVC) continued to have strong growth. For 1996, the two HDL conferences (VIUF for VHDL, and IVC for Verilog) announced they would co-locate [40]. By 1998, they were combined into a true joint conference [41], and in 1999 it was renamed the 8th Annual HDL Conference and Exhibition (HDLCon) [42]. In 2003, the name changed again to Design and Verification Conference and Exhibition (DVCon) [43]. In 2014, the conference expanded internationally adding DVCon India and DVCon Europe [44][45].

In 1997 Synopsys acquired Viewlogic, which had itself acquired Chronologic in 1994 along with its groundbreaking compiled Verilog simulator VCS (Verilog Compiler Simulator) [46]. Now with an industry-leading Verilog simulator of their own, Synopsys supported both languages across their product line.

In February 2000, the two language groups VHDL International (VI) and Open Verilog International (OVI) merged to form Accellera [47].

The 2001 IEEE Verilog standard added many new features to the language, several of which were inspired by similar VHDL capabilities [48].

Figure 2 again shows the n-gram plot of how often the words VHDL and Verilog appear in the Google Books corpus, now including data extending out to 2000. The number of occurrences of VHDL declined after 1995, while Verilog continued to grow.

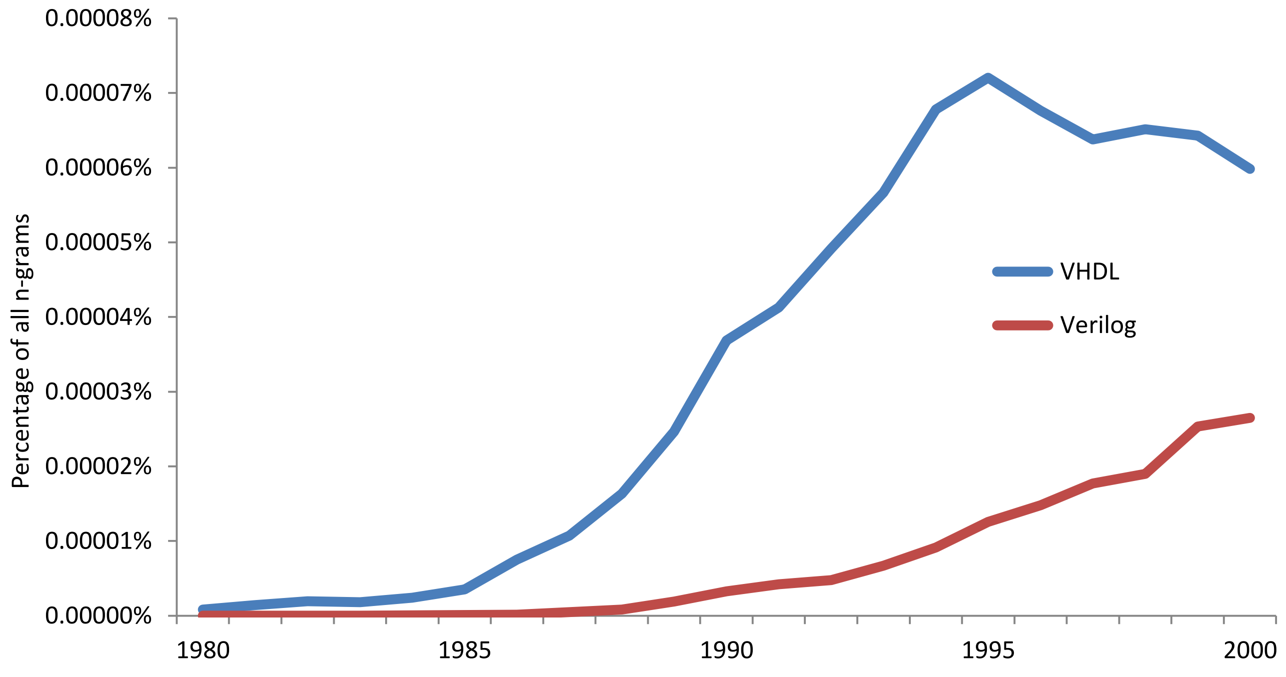

Figure 2: n-gram plot of VHDL and Verilog appearances in Google Books corpus

Was VHDL popularity really declining? Had Verilog won the language war?

3. HDLs in the 21st Century

3.1 C for Hardware Design

Beginning in the late 1990s, several proposals and tools were introduced using C++ with class libraries for hardware design, for example, SystemC and SpecC. These were touted as allowing RTL modeling as well as higher levels of abstraction and system-level modeling. Furthermore, the expectation is that these tools would run on inexpensive commodity servers. Similarly, Java was suggested as a hardware design language [22].

Hardware designers were uniformly unimpressed [49] and eventually these C++ tools were redirected towards system architects and transaction-level modeling.

3.2 The Rise of Verification Languages

As chips began to exceed one million gates in size, the use of HDLs such as Verilog and VHDL for verification (i.e., writing test benches) became unwieldy. A more modern object-oriented approach was required [50].

As a result, a number of hardware verification languages were introduced, for example e and Vera. Also, other system-level tools such as Rosetta and PSL make their appearance [1].

3.3 Verilog and SystemVerilog

In 1997 Co-Design Automation was founded with the intent of developing a new simulator and an entirely new language suitable for hardware design, verification, and software development. Rather than create a new language, Co-Design soon decided to enhance and extend Verilog. By 1999, the new language Superlog was introduced, along with a simulator SYSTEMSIM as well as a translator SYSTEMEX that output traditional Verilog [22].

By early 2001, the EDA world settled into two groups: one group advocating the C++ SystemC, and the other group advocating, "evolve Verilog" into Superlog. Work began by Accellera to standardize Superlog[4]. In May 2002, this new language extension was approved and became known as SystemVerilog[5] [22].

Accellera continued its work on the language, culminating in SystemVerilog 3.1a in 2004. Further standards work was undertaken by the IEEE resulting in the "IEEE Standard for SystemVerilog—Unified Hardware Design, Specification, and Verification Language" approved in November 2005 [51]. A corresponding update to the Verilog standard maintained consistency of the base Verilog language with the new SystemVerilog standard [52].

In 2009, the two standards were merged into a single standard and a single language: SystemVerilog [53]. Further enhancements and errata corrections resulted in the most recent version in 2012 [54].

Synopsys acquired Co-Design in late 2002 [22]. Synopsys became a major advocate for SystemVerilog. At DVCon '03 Synopsys CEO Aart de Geus gave the keynote address and strongly supported SystemVerilog as the one language for the future. Legacy Verilog code is compatible with

[4] The Accellera committee originally called this next generation language *Verilog++* [50].
[5] The first release of the SystemVerilog standard by Accellera was given the version number "3.0". This made it clear that SystemVerilog was the third generation of the Verilog language (IEEE 1364-1995 being the first generation of Verilog or 1.0, and 1364-2001 being the second generation, or 2.0) [50].

SystemVerilog and can still be used moving forward, but the implication was that VHDL had no future [55]. This caused much consternation and spirited discussion in the industry [56][57]. Was VHDL dead?

3.4 VHDL

Between 1993 and 2008, the IEEE VHDL standard did not change significantly. In 2000, protected types were introduced [58], and in 2002 only minor changes were made [59].

In sharp contrast, the 2008 VHDL standard [60] added a large number of enhancements and improvements, making VHDL more suitable as a verification language, and notably incorporating many features first seen in Verilog and SystemVerilog. For example: external names, force and release, generate enhancements, sensitivity list updates, and the VHPI interface to C [61]. Many syntax changes were included to reduce VHDL's verbosity [62]. Despite the lack of object-oriented capabilities, new packages were developed that greatly increase VHDL's usefulness as a verification language [63].

4. Languages in Today's Flows

Why do we care about languages in flows? It turns out that which language you choose can have a very large impact on your design flow, for both simulation and synthesis. Different vendors have different levels of support for VHDL and SystemVerilog. It is important to be aware of what these are when choosing a language (if you should be so lucky as to get a choice), and especially when choosing a full set of tools to support your flow. Choosing the right path from the beginning can eliminate a lot of problems further down the road, and you can only do that with the right information.

Before we can talk about flows, we need to get some definitions out of the way.

4.1 HDL versus RTL

A lot of people use these two terms interchangeably, but they do not mean the same thing. HDL stands for *Hardware Description Language*, which is a full programming language. It is highly configurable, and allows coding of abstract structures. For example, dynamically linked lists or anything with "dynamic" or "variable" in the name. RTL is a coding style called *Register Transfer Level*, which can be written in an HDL. But RTL is not a language, it is just a coding style, and when using it you are limited to a well-defined subset of the HDL's capabilities.

In this paper, we casually talk about "Verilog" and "VHDL" as if they mean the same thing to everyone, but they don't. An implementation engineer is going to code things very differently from a verification engineer.

Another way to distinguish between HDL and RTL is to consider what tool you are targeting. Simulators are really smart and can process many different coding styles, regardless of the HDL being used. However, not all EDA tools are that smart, and it is important to code to the "dumbest" tool in your arsenal. No one cares how compact, elegant, or sleek your code is if it breaks the tools!

4.2 How Many HDLs Do We Have Today? Two, Three, More?

First of all, let's clear something up. A lot of people refer to Verilog and SystemVerilog as two different languages, but they are not. Hopefully reading about the history in Section 3. of this paper made it clear that SystemVerilog is an evolution of earlier Verilog specifications. From this point on, we will use "SystemVerilog" to refer to the modern language, and "Verilog-200*" or sometimes just

"Verilog" to refer to prior versions of the language.

Given that, how many HDLs do we have to work with? The answer, as with so many things, is "it depends". To explain why, we need to divide our chip design flow into two parts: *Implementation* and *Verification*. We'll go into detail about each of these in the next sections. But first, let's talk about the "Big Two" languages for a bit longer.

4.3 Some Language Quirks

No language is perfect, and VHDL and Verilog are no exception. Shannon Hilbert created a nice online chart of VHDL and Verilog pros and cons [77] and here are some of the ones that stand out:

VHDL

- Has no `` `ifdef `` construct — requires another complete architecture
- Does not support hierarchical referencing (although this was added in VHDL-2008)
- File reading is order-dependent
- More likely to require passing in generics (parameters) which can be error-prone
- Case insensitivity can causes issues
- Is a strongly-typed language, which can be both good and bad

Verilog-2001/Verilog-2005

- Compact
- Is a weakly-typed language, which can be both good and bad
- File reading is not order-dependent
- No support of custom or enumerated types
- Closer to actual hardware — can instantiate gates

SystemVerilog

Wait; weren't we only going to talk about the "Big Two"? But since SystemVerilog is what is current, it is worth talking about separately, if only to highlight where it differentiates itself from older standards like Verilog-2001. SystemVerilog incorporates some of the things that were missing. In fact, one of the original goals of SystemVerilog was to incorporate the strengths of VHDL. You could even say that VHDL lives on in SystemVerilog. So what is different about it [64][65]?

- Adds declaration packages and classes, which can be order dependent
- Is a superset of Verilog-2001 and Verilog-2005
- Incorporates many of the strengths of VHDL such as (to name a very few):
  - Enumerated types
  - Multidimensional arrays
- Implements RTL-specific structures like *always_latch*, *always_ff*, *always_comb*
- Uses a less verbose syntax with stronger typing
  - For example, implicit port connections with *.** and *.name*
- Allows directly calling C functions
- Brings significant improvements geared toward verification

4.4 Writing Mixed RTL

It is important to remember that you can write good or bad RTL in any language. Since we are assuming you want good RTL, let's figure out how that happens.

If you are reading this paper, then chances are you know some coding languages. In fact, you probably know a lot about either Verilog/SystemVerilog or VHDL. But most of us would not describe ourselves as an expert in both languages. That means in a mixed-language environment you need to be careful to include two different sets of experts when selecting your team. Most companies lean either toward VHDL or toward SystemVerilog, so finding someone to support the "other" language in your corporate culture can be a challenge. In some cases the challenge is a single person espousing their love for VHDL, but in other cases it is a question of exposure. People will gravitate to the language they know. That's where training can help.

4.5 Training

Since most engineers coming out of college have a basic knowledge of either SystemVerilog or VHDL, or at least the synthesizable subset of them, they tend not to need RTL training. In addition, the lack of evolution of the synthesizable subset of SystemVerilog means there is less need for training in writing RTL. Training in RTL is only needed when switching languages, and this is often from VHDL to SystemVerilog.

Many engineers welcome the chance to learn a new skill, while others are dragged kicking and screaming to learn SystemVerilog. According to Stu Sutherland, expert HDL instructor at Sutherland HDL, the majority of engineers see learning SystemVerilog, on both the design side and the verification side, as opening doors for their career future. Stu says:

> Frequently, VHDL engineers start my class bitterly complaining about SystemVerilog's loosely typed language rules, and how those rules let engineers, especially designers, make stupid mistakes. Almost universally, however, by the end of the class and after completing several labs using SystemVerilog, those same engineers have begun taking advantage of those loosely typed rules. I can't begin to count how many times I hear an epiphany yelled out during a lab along the lines of "Now I understand why so many engineers like this language!"

4.6 Pros and Cons of Using Multiple HDLs for RTL

Most companies today use a mixture of VHDL and SystemVerilog for their RTL, whether or not they are creating new content in both languages. If you are creating new content in both languages, you have the burden of needing expertise in both languages; but even if you are using tested IP, you still need to have enough of an understanding of the language that IP is written in to be able to debug any issues with it.

Pros:

- Have access to a larger set of IP
- No translations needed
- Each block of a design can be modeled in the HDL that best represents its functionality

Cons:

- Design teams need to be able to understand and debug multiple languages
- EDA tools being used need to support multiple languages
- Can require multiple software licenses at an additional cost

Even if you try to use a single language for your design, if that language is VHDL you have already lost that battle. Today, most ASIC design infrastructure is in SystemVerilog, including libraries for both standard cells and memories, and of course, netlists. VITAL (VHDL Initiative Towards ASIC Libraries, also an IEEE standard 1076.4) was launched at DAC in 1992 to provide a way to capture libraries in VHDL. The general feeling in the industry was that not having ASIC libraries available in a VHDL format was the single most important factor limiting the use of VHDL in design [66]. VITAL, while still supported by some EDA companies and vendors, never took off. The latest revision of the IEEE standard for VITAL was in 2000; more than a lifetime ago in the ASIC design world, and as of 2009 this standard has been withdrawn.

As for netlists, while older EDA companies like Synopsys [67] and Cadence [68] still support writing and reading VHDL netlists, newer EDA companies haven't bothered adding the same support. Mentor Graphics' Oasys synthesis tool only supports the `write_verilog` command [69], and a quick search of the ATopTech documents provided no hits whatsoever for "VHDL" [70].

Finally, some see FPGAs and emulation as the last bastion of VHDL. This has traditionally been the case, however SystemVerilog use in FPGAs is growing [71].

4.7 Maintaining Mixed RTL

There is a misconception that once a design "works on silicon" the RTL never needs to be revisited. This is wrong for several reasons. First of all, the languages change over time. For example, the syntax of the `generate` statement changed between Verilog-2001 and Verilog-2005. What was once legal became illegal and code changes were required.

Verilog-2001	**Verilog-2005**
`// correct in Verilog-2001`	`// correct in Verilog-2005`
`// but incorrect in Verilog-2005`	
`genvar i;`	`genvar i;`
`generate              // required in Verilog-2001`	`generate   // optional in Verilog-2005`
`  begin: gen_block // illegal in Verilog-2005`	
`    ...`	`...`
`    for(i=0;i<10;i=i+1)`	`for(i=0;i<10;i=i+1)`
`    begin : gen_block1`	`  begin: gen_block1`
`      ...`	`    ...`
`    end`	`  end`
`  end`	
`endgenerate           // required in Verilog-2001`	`endgenerate // optional in Verilog-05`

Another example is the change in the vector size of the "integer" and unsized literals between Verilog-2001 and SystemVerilog. In Verilog-2001 the size was left up to the software tool, and in SystemVerilog the size is exactly 32 bits.

In addition to the language itself changing, the way tools interpret the language can change. A few years ago, Synopsys introduced a new variable called `compile_enhanced_resource_sharing` that caused very different structures to be synthesized. While very efficient, this new interpretation of the RTL caused logic equivalence tools to flounder. The easiest fix would have been to turn off this variable, but that would leave Quality of Results (QoR) on the table. The next best fix was to recode the RTL to change the way resource sharing could happen. Sometimes changes to RTL models are required to keep up with EDA tool improvements.

Even for IP that is "never going to change", there is a challenge to maintaining it in two different languages.

5. Implementation Challenges of Mixed Language Designs

Implementation is all about our RTL: the nice, neat subset of HDL that becomes our AND gates and OR gates. So in order to implement a design, what do we need to do?

5.1 Reading in the RTL

In general, reading in a Verilog-200* design can be done in any order. There is one caveat to this, and that is if there are missing include files such that the Verilog defines are inherited between files. Fortunately, this is an easy problem to fix and monitor by adding the appropriate include statement to each file, so reading in a pure Verilog-200* design remains quite simple.

SystemVerilog is a little bit more difficult in that all of the packages need to be read in first. A simple rule that requires packages to be defined in separate files makes this easy to accomplish.

Then there's VHDL, which can be more complicated with respect to order-dependence. In VHDL, packages also need to be read in first, so that's no different. But then these are followed by entities (which define the name and input/output ports of a module), and then architectures (which contain the guts of the module). For synthesis there needs to be a one-to-one assignment of architectures to entities, but there is no requirement they be in the same file. For configurable code, they often are not, and you have to make sure you are getting the correct combinations of files. Libraries add an extra level of complication, as each library needs to be analyzed before it can be referenced. All of this order dependence in VHDL requires that the RTL designer provide additional information for the implementation engineer, and provides another avenue for miscommunication (i.e.: simulation-synthesis mismatches, or bugs).

5.2 Synthesis and Physical Synthesis

Let's go ahead and tackle the big one next. We all know we don't have a design unless we can synthesize it. We also know that we cannot synthesize just anything; it needs to be good RTL and not just something written in an HDL.

For example, this code is very versatile:

```
assign  o_data = ( s_sel >= SEL_MIN && s_sel <= SEL_MAX ) ?
     i_data[(s_sel - SEL_MIN)*DATA_BITS +: O_TAPS*DATA_BITS] :
     {O_TAPS*DATA_BITS{1'b0}};
```

But the selector value here requires a huge number of muxes to calculate, blowing up the size of the design. In order to figure this out, it was very important to be able to read and understand the Verilog code. The synthesis engineer was able to localize the issue and explain the problem to the RTL designer only because she was familiar with SystemVerilog. If this design had been written in VHDL then she may not have been able to do that.

Once the problem was pointed out, the RTL designer was able to simplify the code. He used his a priori knowledge to remove some of the configurability from the `s_sel` signal and the new code synthesized quite nicely[6]:

```
always @*
   case ( s_sel )
     -1       : o_data = i_data[  0 * DATA_BITS +: O_TAPS*DATA_BITS ];
      0       : o_data = i_data[  1 * DATA_BITS +: O_TAPS*DATA_BITS ];
      1       : o_data = i_data[  2 * DATA_BITS +: O_TAPS*DATA_BITS ];
      default : o_data = {O_TAPS*DATA_BITS{1'b0}};
   endcase
```

Physical synthesis is a little different than plain old "synthesis". In physical synthesis, it is important to keep the ties to the RTL in the database in order for the optimization routines to work properly. That means all IP needs to be delivered as RTL, not as a synthesized or even an elaborated netlist. Again, the implementation engineers who are working in a multiple language environment can no longer completely distance themselves from the RTL[7].

Getting synthesis to work on your mixed-RTL design is a huge step, but you can't tape out that synthesized design unless a bunch of other stuff works, too—stuff that has to work on RTL. The following sections give just a few examples.

5.3 Logic Equivalence Checking

Logic Equivalence Checking (LEC) is meant to check that the simulation results match the synthesis results. In other words, the behavior of the code you're simulating is identical to how the gates will function in silicon.

Having order-dependent RTL poses an issue in the implementation flow. Both VHDL and SystemVerilog can be written in an order-dependent way, but as we have mentioned before, VHDL is much more prone to order dependence. This can make Logic Equivalence Checking difficult. In order to get matching results with order-dependent code, it is common to use the identical script to read the RTL into your LEC tool as is used for the synthesis tool. But unless that script is also used for simulation, you may not be checking what you think you are checking. Since implementation and verification tasks are generally quite separate, this can be a big gap in the design verification process. Using correctly coded SystemVerilog with limited order dependence (i.e., packages only) instead minimizes this gap.

5.4 DFT Tools

DFT (Design for Test) can be inserted at different times during the design process. In some flows, test logic is inserted after the netlist is optimized, so the input to the DFT tool is a netlist. In other flows, the test logic is inserted directly into the RTL before any synthesis has occurred.

Inserting DFT hardware at the RTL level can have many advantages, including getting optimum

[6] In addition to illustrating why working with mixed languages is difficult for implementation, this example also shows why it's important to code with your hardware in mind, using the proper RTL coding style.

[7] Why does physical synthesis need to have links to the RTL? In general physical synthesis tools are the next generation of synthesis. Their algorithms work at a higher level of abstraction, so they want to see "+" instead of a group of gates that represent, say, a ripple-carry adder. When doing placement, it is important to be able to go back to the implementation selection stage in order to get optimal results for timing, congestion, and other components of the synthesis cost function. This can only be done if the links to the RTL are intact.

synthesis QoR, and avoiding issues using a third-party DFT tool during physical synthesis. But in order to accomplish this, the DFT tool needs to be able to parse RTL, a much more difficult task than parsing a gate-level netlist.

One Broadcom team uses a third-party tool to insert LBIST (Logic Built-in Self-test) and MBIST (Memory Built-in Self-test) hardware. When they first started adding LBIST and MBIST to their Verilog-2001 designs, they were able to insert the hardware at the RTL level with no problems. However, as this team started to include Verilog-2005 and SystemVerilog constructs in their RTL, they found that the DFT tool was no longer able to parse the RTL, and they were forced to push the DFT hardware insertion to the synthesized netlist. This not only cost QoR as the DFT hardware was not as optimized as it could have been, it forced them away from using physical synthesis methodologies as they had to insert the third-party tool into the flow (see Section 5.2 discussion of physical synthesis). The DFT tool vendor has since resolved this problem, but in the meantime it affected many designs.

This example shows how the different EDA tools in a flow can interact and force implementation flow decisions to be made. In this case, the tool could not interpret new features added to a single language. The impact is increased when you consider that EDA companies need to support multiple languages.

5.5 Homegrown Tools

You've seen these, you may even have written some. Homegrown tools may seem like the panacea to EDA tools' shortcomings, but unfortunately that is only true for a short time. There's a reason EDA companies continue to make money, and that is because it is hard to keep your tools up-to-date with all of the latest standards, especially when that's your side job.

Once a homegrown tool makes it into your flow it can be very hard to get rid of, and if that tool is old you may find yourself limited in what you can do with languages. For example, you may be relegated to using only Verilog-2001 constructs to code a function that really requires the features of SystemVerilog. It is a waste of time, and will potentially result in a lower-quality design. In this case your company is supporting more than two languages: Verilog-2001, SystemVerilog, and (potentially) VHDL, too.

The best thing to do here is to cut your losses. If you can't find a commercially available tool that supports what you need to do, you may need to ask yourself why you're doing it. Our value-added as engineers is not using obscure, out-of-date Verilog-2001 constructs to build a modern chip, it is to design and implement new, cutting-edge functions and get our chips to market as soon as possible so they can make money.

5.6 And More…

We can't possibly have covered all of the tools and tasks that need to read RTL here. There are LINT checkers, constraint analysis tools, clock-domain-crossing tools, design restructuring tools, and more, all of which need to read designs in RTL format. Many tools might use the same front end, like Verific, which would minimize parsing issues. Making sure every tool in your flow supports both SystemVerilog (or whatever Verilog standard you are using) and VHDL competently is a challenge that could be eliminated by settling on a single language flow.

5.7 You May Never Have Noticed Any of This… Yet

How? Even if you're an implementation engineer, you may never have had to start from RTL. If your expertise is timing closure, or PNR, your work hasn't started until all of this RTL work is complete.

But things are changing:

- Constraints development now starts at RTL
- Logic Equivalence Checking requires a path back to the RTL
- Getting good QoR from physical synthesis tools requires maintaining a connection to the RTL
- DFT bugs can require RTL changes

Don't be surprised if you are asked to look at RTL more and more, and then you too can enjoy the challenges of working in a mixed-RTL language environment!

6. Verification Challenges

While the last two sections have highlighted some of the challenges of working in a mixed language environment for RTL, clearly it *is* possible to have a mixed language environment for RTL. SystemVerilog and VHDL co-exist, albeit sometimes contentiously, and chip development goes on.

If the language wars in the 90s were about RTL, then today's language wars are about verification. Verification can be done with any number of HDLs, each with their own strengths and weaknesses. We can come up with this approximate timeline [72]:

- Late 1980s: HDL training for RTL coding begins
- Late 1990s: HDL training for RTL coding peaks
- 2000: Training for SystemC for verification begins (pushed by Synopsys)
- Late 1990s – early 2000s: Vera and e
- 2003: SystemVerilog starts – wave is still going
- 2005: Training for SystemC for verification crests
- 2006: VMM[8] using SystemVerilog base class libraries begins
- 2008: OVM[9] starts
- 2010: UVM[10] starts – that's the big wave
- ca. 2012: VMM and OVM usage peaks

6.1 Training – Revisited

For some, the trend here is away from training in HDLs for RTL, and toward training in languages for verification. With the advent of UVM to SystemVerilog, it is now possible to write an extremely powerful testbench in SystemVerilog, which explains the huge increase in UVM's use. Since most graduating engineers have little to no knowledge of UVM, and since UVM is still rapidly evolving, training is required. This correlates with input from a Broadcom engineer who said that he doesn't even ask any more if someone has experience in VHDL or SystemVerilog, but rather he asks if they know UVM, as that's harder to learn and fewer people know it.

Currently at training provider Willamette HDL, 95% of their training is focused on SystemVerilog with UVM, and the trend is definitely away from VHDL and towards Verilog in general [72]. Even customers who use VHDL for RTL are moving towards SystemVerilog for verification. And once you're using SystemVerilog for verification, you don't care what language the design/RTL is in, thanks to SystemVerilog's `bind` feature [73].

[8] VMM (Verification Methodology Manual)
[9] OVM (Open Verification Methodology), a SystemVerilog library
[10] UVM (Universal Verification Methodology), also a SystemVerilog library

That is not the case at another training firm, Sutherland HDL, where about 30% of their training is still on RTL, much of which is for engineers transitioning from VHDL to SystemVerilog, for both design and synthesis [50].

Even though these two training companies see differences as to whether training is for RTL or for verification, both clearly see a trend away from VHDL training and toward SystemVerilog training. This is not necessarily indicative of total language use, as it gives no insight into language use by well-trained teams that are already ensconced in their language of choice, but in our opinion it is an indicator of the future direction of language use.

6.2 VHDL-2008

What about VHDL-2008, you may ask? During development, VHDL-2008 was sometimes referred to as *SystemVHDL*, and for the most part is playing catch-up with SystemVerilog. This is clear by looking at its history. Not much had changed through all of the IEEE revisions of VHDL until 2008, at which point there were some very significant changes, most of which came from SystemVerilog. It's as if the VHDL committee noticed at the last minute they were losing the war, and decided to try to come up to speed. Today VHDL-2008 is considered more of a verification language, but it is a case of too little, too late, and it has failed to get a toehold in EDA industry support. Besides, why are we maintaining two languages to do the same thing?

The previous section of this paper went over the proliferation of mixed-language design. As we mentioned before, SystemVerilog testbenches are able to handle mixed languages, but VHDL testbenches, even VHDL-2008 testbenches, cannot. The only people using VHDL as their verification language are people who only use VHDL [74], and this is part of a rapidly shrinking group [75].

6.3 Verification IP

Another way to measure the dominance of VHDL-2008 versus SystemVerilog in the verification arena would be to survey the availability of Verification IP, or VIP. This would be an indication of the future of verification languages from the perspective of the companies that write, sell, and maintain VIP code. We speculate that this would show the same trends toward SystemVerilog that we see in the rest of the industry.

7. EDA Infrastructure

There is a saying that goes, "If it's not tested, it's broken"[11]. The less of anything an EDA tool sees, the less well it will able to support that thing. Given the decline in the use of VHDL, is it any surprise that EDA companies are dedicating fewer of their resources to supporting it?

7.1 Coding Styles in General

It is important to keep EDA tools in mind when writing your code. Why? Our end product (and where we make our money) is silicon, not RTL. In order to make that silicon, the RTL has to be processed successfully by a variety of EDA tools.

[11] Landman's Law (circa RISC I tapeout 1980–81): For any sufficiently large chip, if there is a type of error for which you have no automated check, your chip will contain at least one instance of that error.

PROOF: Since the chance of the error occurring in a fixed size chunk of the chip is non-zero, the chance of it not occurring anywhere decays exponentially towards zero as a function of chip size.

Landman's Lemma (a few years later, maybe 1986ish): All chips are now sufficiently large [78].

When coding, think about the coding styles that your EDA tool has been exposed to, what the vendor will have tested, and what is most likely to work. Which language is that?

7.2 Support for New Features

Anecdotal evidence from an engineer at Broadcom suggests that if a construct exists only in VHDL, EDA vendors are not likely to enhance their tools to support it. But if that construct exists in Verilog or SystemVerilog, they will. Of course, if you ask the vendor directly about their mixed-language support, they'll say it's a non-issue.

This is supported by sources at the Synopsys Support Center, who say that:

- There are a low percentage of people using VHDL.
- We see few "full" VHDL designs in the U.S.
- Some of the VHDL designs we see are when customers use IP that is in VHDL.
- There is more VHDL use in the UK and other parts of Europe than in the US. We also see some use of VHDL in India.

Additionally, VHDL-specific support tickets comprise a low percentage of all tickets. There are few full-VHDL designs in the US, and what is in VHDL is often IP that is no longer being actively updated. Even in Europe, long known as VHDL-centric, things are moving toward SystemVerilog instead.

7.3 Case Study: Formality

We met with the Formality R&D team and asked them some questions about HDL support [76]. According to them, there is a standing committee that meets weekly to discuss new Verilog features, but no such committee exists for VHDL. In fact, over the previous five years (2010–2015) only two new features have been added to support the VHDL-2008 standard.

As for the Formality customer base, two-thirds is Verilog (including SystemVerilog), which is mirrored by the support burden. They report that no new RTL code is being written in VHDL, and no one is asking for new features in VHDL, either. They also claim that the Formality team is more interested in VHDL than the Design Compiler (DC) team is, which bodes badly for VHDL support in DC!

In general, the feeling of the Formality team is that SystemVerilog readers are much more robust than VHDL readers, and even in Europe, where the bulk of VHDL is written today, there is a huge migration to SystemVerilog.

Since our discussion with the Formality R&D team, we've had the opportunity/burden of working with an implementation team on a very large VHDL block for one of Broadcom's chips, and we experienced the pain of being in the minority first-hand.

On this project there were five very large blocks written in VHDL. Of the five, Formality could only read in three due to some sort of VHDL syntax issue. We were unable to debug this issue with code snippets, and were unable to provide the entire design to Synopsys for debug, so we ended up not using Formality on those blocks. Note that Design Compiler and Cadence's LEC were both able to read in these designs, however SpyGlass also had some issues.

For the three designs that Formality *was* able to read, there was one bug with an incorrectly issued FMR_VHDL-153 error that prevented the tool from running[12]. There were also quite a few other FMR_VHDL messages generated where debug was difficult as no documentation seems to exist, for

[12] Fixed in FM 2014.09-SP5.

example for FMR_VHDL-060[13]. For comparison, we also work on a lot of Verilog designs (including SystemVerilog) and there are very few messages that are not documented.

7.4 Case Study: Verific

Looking at another EDA vendor, the opinion at Verific Design Automation is that VHDL is never going away completely, even though the trend is downward. Verific, provider of SystemVerilog and VHDL parsers to more than 60 EDA applications, shows distribution of their licensees as follows:

	2010	**2015**
VHDL	70%	66%
SystemVerilog	96%	100%

8. Conclusions and Recommendations

Where are we now, and why should I care?

We are well into the second decade of the 21st century. Many predicted we would already be down to one HDL at this point, but we aren't. Supporting multiple HDLs for implementation can be a pain, but it seems to be getting easier, in part because fewer people are using VHDL [75], and even fewer are creating new RTL in VHDL.

Who *is* still using VHDL? In 2015, the US appeared to be 80–90% Verilog/SystemVerilog. There are certain US military and aerospace vendors continuing to use VHDL, despite the lack of a DoD mandate. Europe used to be a huge VHDL supporter, but this is a legacy issue now and there is very little new VHDL being written.[14] There's little VHDL usage in India/Asia, as these are historically Verilog. When VHDL was mentioned to the Broadcom team in Israel, there were chuckles.

The remaining big holdout for VHDL is FPGA designers. It seems VHDL-1993 has been sufficient to meet their needs for design and verification, so there has not been a pressing need to change. FPGA size and complexity has always lagged ASICs by a few years. However the increasing verification requirements for today's more-and-more complex FPGAs are encouraging FPGA designers to transition to SystemVerilog, first for their testbench and then for the full design flow [75].

Even people who are using VHDL for RTL implementation are beginning to use SystemVerilog as a verification language [74][75]. Several companies offer training classes for Verilog and SystemVerilog, and many of these companies offer classes specifically for VHDL designers transitioning to SystemVerilog[15]. Once engineers are familiar with SystemVerilog for verification, it is a small jump to code the hardware in SystemVerilog too, or so we can hope.

[13] Still nothing in SolvNet as of January 2016.
[14] Except for France where they cling to VHDL.
[15] However there seem to be no classes going the other way!

What do you think is the most popular IEEE standard for EDA? Which ones are purchased most often? And which ones show the least interest? In June 2011 the IEEE Standards Store ranked sales as follows [79]:

Rank	Standard	Name
1	1364-2001	Verilog Hardware Description Language
2	1800-2009	SystemVerilog–Unified Hardware Design, Specification, and Verification Language
3	1076-2002	VHDL Language Reference Manual
4	1076-1993	VHDL Language Reference Manual
…		
6	1364-1995	Hardware Description Language Based on the Verilog Hardware Description Language
7	1800-2005	SystemVerilog: Unified Hardware Design, Specification and Verification Language
…		
23	1076-1987	VHDL Language Reference Manual
…		
27	1076/INT-1991	Interpretations: IEEE Std 1076-1987, IEEE Standard VHDL Language Reference Manual
…		
32	1666-2005	SystemC Language Reference Manual
…		
34	1364-2005	Verilog Hardware Description Language

In this table we've included only the main HDLs. Did you notice what is missing? The newest VHDL standard 1076-2008 *is not anywhere in the top 36*.

Speaking of standards, here's a telling difference between VHDL and SystemVerilog. The IEEE working group for SystemVerilog is *entity based*, whereas the VHDL working group is comprised of *individuals* acting on their own. Thus each change to the SystemVerilog standard is tacitly approved by the EDA companies as something they will support and implement, while no such guarantee exists for updates to the VHDL standard[16]. The fact that few EDA companies have implemented all of the new features in VHDL-2008 is evidence that the EDA companies do not really approve of the standard.

[16] One downside of the entity based balloting is the increase in politicization of what should be a straightforward engineering process. EDA vendors can be reluctant to approve changes in the standard unless they see a way to make a profit. Nevertheless, the fact remains that if a change is approved, it already has the support of the major EDA vendors.

If you need further proof of the decline of VHDL, Figure 3 shows another n-gram plot of how often the words VHDL, Verilog, and SystemVerilog appear in the Google Books corpus, now with the data extending out to 2008. The number of occurrences of VHDL has been declining since 1995, while Verilog and SystemVerilog are increasing in popularity.

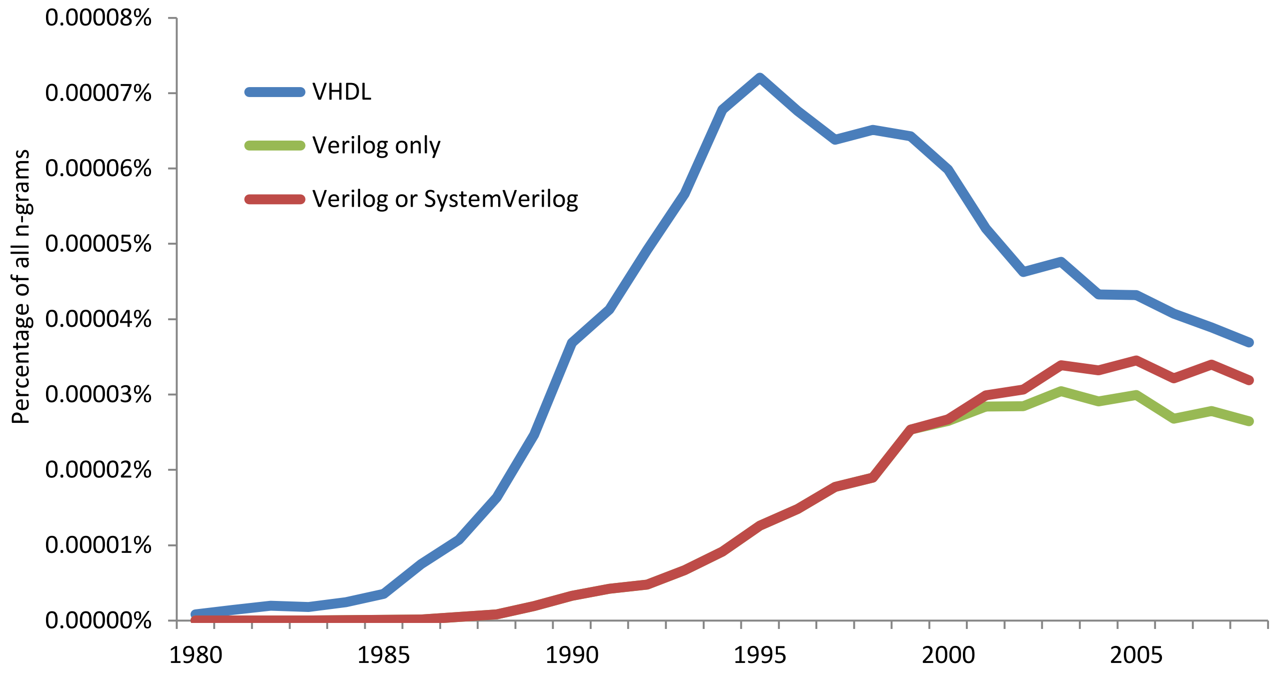

Figure 3: n-gram plot of VHDL, Verilog, and SystemVerilog appearances in Google Books corpus

At some point it will become counterproductive for EDA vendors to support VHDL along with SystemVerilog. We are starting to see that already. For example, which vendors offer *complete* support of the VHDL-2008 standard, across *all* of their tools? None that we know of.

All of these factors mean that your world as a design engineer is shifting. It matters to you because it's going to affect your job. If you only know VHDL, you will have to change, and if you have been supporting both VHDL and SystemVerilog, your world is getting better. But don't worry, some new challenge[17] will come fill the void left by not having to support two languages! Things are only getting more interesting, not less.

[17] Such as, arguing about UVM versus OVM for verification. Or perhaps debating CPF versus UPF for power. There's always another war…

8.1 And the Winner Is…

VHDL is a wonderful, powerful, expressive language, with a long and storied history. However, it did not win the HDL language war. Yes, there are features of VHDL that are not available in SystemVerilog. But these are not deal breakers. Anything you can express in VHDL, you can also express in SystemVerilog.

We can make a couple of analogies. Verilog is like the English of HDLs: everything you like becomes a part of it, and it is spoken around the world. VHDL is more like the Latin of HDLs: we learned many good things from it, but it is old and it is dead, and people do not speak it anymore.

In his 1995 IVC keynote, Joe Costello, then CEO of Cadence, famously said [80]:

> VHDL is one of the biggest mistakes the Electronics Design Automation industry has ever made. A $400 million mistake. Wouldn't this money have been better spent on handling submicron design, testability issues, or even a new type of HDL that had significantly more capabilities than what Verilog and VHDL offer today?

Today, over twenty years later, we have just such a new type of HDL—and it is called SystemVerilog.

VHDL will be with us for a long time to come and we will always have issues supporting legacy VHDL code. But if we can stop writing more we can get to a place where all of the legacy code is well handled by the tools, and we can focus on supporting a single language, the winner, SystemVerilog.

9. Acknowledgements

Grateful thanks to the following for their input and review: Bertrand Boussert, Victor Grimblatt, Bob Hauser, Don Mills, Asim Naseem, the Formality R&D Team, Michiel Ligthart, Kurt Baty, Mike Baird, Cliff Cummings, John Sanguinetti, Howard A. Landman, and several engineers who prefer to remain unnamed. Please note they spoke with us as individuals and were not representing their employers.

Major thanks to our SNUG reviewer, Stuart Sutherland. His thoughtful comments and insights greatly improved this paper. Thanks, Stu.

As always, these are our thoughts only.

10. References

[1] Stephen A. Edwards, “Design and Verification Languages,” in *EDA for IC system design, verification, and testing*, Louis Scheffer, Luciano Lavagno, and Grant Martin, eds. CRC Press, 2006.

[2] G. Jack Lipovski, "Hardware description languages: Voices from the Tower of Babel," *IEEE Computer,* vol. 10, no. 6, pp. 14-17, June 1977.

[3] William M. vanCleemput, "Computer hardware description languages and their applications," in *Proc. 16th Design Automation Conf.*, 1979, pp. 554-560.

[4] Claude Elwood Shannon, “A symbolic analysis of relay and switching circuits,” M.S. thesis, Dept. of Elect. Eng., MIT, 1940.

[5] Arthur W. Burks, Herman H. Goldstine, and John von Neumann, "Preliminary discussion of the logical design of an electronic computing instrument," report prepared for U. S. Army Ordnance Dept., 1946, in *John von Neumann Collected Works: Volume V-Design of Computers, Theory of Automata and Numerical Analysis*, John von Neumann and A. H. Taub, eds. Pergamon Press, 1963.

[6] Irving S. Reed, "Symbolic synthesis of digital computers," in *Proc. 1952 ACM Nat. Meeting,* (Toronto), pp. 90-94.

[7] Irving S. Reed, “Symbolic design techniques applied to a generalized computer,” MIT Lincoln Laboratory, Tech. Report TR-141, 3 January 1956. Referenced by Irving S. Reed, “Symbolic design techniques applied to a generalized computer,” *IEEE Computer*, vol. 5, no. 3, pp. 46-52, May/June 1972.

[8] “Hardware Description Languages,” special issue, *IEEE Computer*, vol. 7, no. 12, Dec. 1974.

[9] Robert Piloty, Mario Barbacci, Dominique Borrione, Donald Dietmeyer, Fredrick Hill, and Patrick Skelly, "CONLAN: a formal construction method for hardware description languages: basic principles," in *Proc. May 19-22, 1980, National Computer Conference*, pp. 209-217.

[10] Robert Piloty, M. Barbacci, Dominique Borrione, Donald Dietmeyer, F. Hill, and Patrick Skelly, *CONLAN report*. No. 151. Springer Science & Business Media, 1983.

[11] U.S. Congress, Office of Technology Assessment, *Microelectronics Research and Development – A Background Paper*. U.S. Government Printing Office, 1986.

[12] G. W. Preston, *Report of IDA Summer Study on Hardware Description Language*. Institute for Defense Analyses, 1981.

[13] Moe Shahdad, Roger Lipsett, Erich Marschner, and H. Cohen, "VHSIC hardware description language," *IEEE Computer*, vol. 18, no. 2, pp. 94-103, Feb. 1985.

[14] Dominique Borrione, Robert Piloty, Dwight Hill, Karl J. Lieberherr, and Philip Moorby, "Three Decades of HDLs, Part II: Conlan Through Verilog," *IEEE Des. Test. Comput.,* vol. 9, no. 3, pp. 54-63, Sept. 1992.

[15] Yong Chu, F. J. Hill, M. R. Barbacci, G. Ordy, Bryant Johnson, and Mike Roberts, "Three decades of HDLs, Part 1: CDL through TI-HDL," *IEEE Des. Test. Comput.,* vol. 9, no. 2, pp. 69-81, June 1992.

[16] Allen Dewey and Aart J. De Geus, "VHDL: toward a unified view of design," *IEEE Des. Test. Comput.,* vol. 9, no. 2, pp. 8-17, June 1992.

[17] *IEEE Standard VHDL Language Reference Manual*, IEEE Std 1076-1987.

[18] *Standard General Requirements for Electronic Equipment*, MIL-STD-454L, U. S. Department of Defense, 20 September 1988.

[19] Michael Carroll, “VHDL—panacea or hype?” *IEEE Spectrum*, June 1993.

[20] *Oral History of Philip Raymond “Phil” Moorby*, Catalog number 102746653, Computer History Museum, 2013.

[21] P. L. Flake, P. R. Moorby, and G. Musgrave, "HILO Mark 2 Hardware Description Language," in *Proc. Fifth Int. Conf. Comput. Hardware Description Languages and their Applications (CHDL)*, Kaiserslautern, Germany, 7–9 Sept. 1981.

[22] Stuart Sutherland, Simon Davidmann, and Peter Flake, *SystemVerilog for Design Second Edition: A Guide to Using SystemVerilog for Hardware Design and Modeling*. Springer Science+Business Media, 2006.

[23] *IEEE Standards Interpretations: IEEE Std 1076-1987, IEEE Standard VHDL Language Reference Manual*. IEEE Std 1076/INT-1991.

[24] "VUG/VIUF Collected Conference Proceedings." [Online]. Available: http://www.eda.org/VIUF_proc/

[25] Michael John Sebastian Smith, *Application-specific integrated circuits*. Addison-Wesley, 1993.

[26] Howard A. Landman, "Logic synthesis at Sun," in *COMPCON Spring'89 Thirty-Fourth IEEE Computer Society International Conference: Intellectual Leverage, Digest of Papers*, 1989, pp. 469-472.

[27] Daniel Nenni and Paul McLellan, *Fabless: The Transformation of the Semiconductor Industry*. SemiWiki.com, 2013.

[28] C. Gordon Bell, "Foreword," in Donald E. Thomas and Philip R. Moorby, *The Verilog Hardware Description Language, Second Edition*. Kluwer Academic Publishers, 1995.

[29] *IEEE Standard VHDL Language Reference Manual*, IEEE Std 1076-1993.

[30] *IEEE Standard for VITAL Application-Specific Integrated Circuit (ASIC) Modeling Specification*, IEEE Std 1076.4-1995.

[31] John Cooley, "INDUSTRY GADFLY: 'From Beirut To Bosnia' + Reader Response," Usenet, comp.lang.vhdl, 13 March 1996. [Online]. Available: https://groups.google.com/d/msg/comp.lang.vhdl/Lfynspa9cnM/r6Bd5KwXejQJ

[32] Mahendra Jain, "The VHDL forecast," *IEEE Spectrum*, June 1993.

[33] "Google Ngram Viewer." [Online]. Available: https://books.google.com/ngrams

[34] *IEEE Standard Hardware Description Language Based on the Verilog Hardware Description Language*, IEEE Std 1364-1995.

[35] John Cooley, "Verilog Won & VHDL Lost? -- You Be The Judge!" Usenet, comp.lang.verilog, 31 May 1995. [Online]. Available: https://groups.google.com/forum/ - !original/comp.lang.verilog/0vMkv1zfRUE/LcnTMM5LE8MJ

[36] John Cooley, "Rorschach Testing 273 Engineers With The Verilog-VHDL Contest," Usenet, comp.arch.fpga, 20 November 1995. [Online]. Available: https://groups.google.com/d/msg/comp.arch.fpga/AHzIcNVTG9E/dC1c94wECR0J

[37] *Military Handbook, General Guidelines for Electronic Equipment*, MIL-HDBK-454, U. S. Department of Defense, 28 April 1995.

[38] *Department of Defense Handbook, General Guidelines for Electronic Equipment*, MIL-HDBK-454 NOTICE 1, U. S. Department of Defense, 28 May 1997.

[39] John Cooley, "INDUSTRY GADFLY: The Fall(ing) VIUF '95," Usenet, comp.arch.fpga, 1 December 1995. [Online]. Available: https://groups.google.com/d/msg/comp.arch.fpga/AHzIcNVTG9E/ah9nIyAHHhwJ

[40] IVC, "About IVC," 1997. [Online]. Available: https://web.archive.org/web/19970414023820/http://www.hdlcon.org/aboutivc.html

[41] IVC/VIUF, "IVC/VIUF HDL Conference," 1998. [Online]. Available: https://web.archive.org/web/19980202191343/http://www.hdlcon.org/toppage.html

[42] HDLCON, "The International HDL Conference and Exhibition," 1999. [Online]. Available: https://web.archive.org/web/19990208004605/http://hdlcon.org/

[43] DVCon, "dvc," 2003. [Online]. Available: https://web.archive.org/web/20030408023416/http://www.hdlcon.org/geninfo.html

[44] Accellera, "Community Newsletter: February 2014." [Online]. Available: http://accellera.org/news/newsletters/2014-february

[45] Accellera, "Community Newsletter: May 2014," [Online]. Available: http://accellera.org/news/newsletters/2014-may

[46] Peggy Aycinena, "John Sanguinetti – A Profile," 2004. [Online]. Available: http://www.aycinena.com/index2/index3/archive/john%20sanguinetti.html

[47] Gabe Moretti, "DVCon is the Primary Design and Verification Conference," 2015. [Online]. Available: http://chipdesignmag.com/sld/moretti/2015/02/20/dvcon-is-the-primary-design-and-verification-conference/

[48] *IEEE Standard Verilog Hardware Description Language*, IEEE Std 1364-2001.

[49] John Cooley, "Designers Hate SystemC, CynLibs, & C-based HW Design," SNUG 01 Item 14, 28 March 2001. [Online]. Available: http://www.deepchip.com/items/snug01-14.html

[50] Stuart Sutherland, private communication.

[51] *IEEE Standard for SystemVerilog—Unified Hardware Design, Specification, and Verification Language*, IEEE Std 1800-2005.

[52] *IEEE Standard Verilog Hardware Description Language*, IEEE Std 1364-2005.

[53] *IEEE Standard for SystemVerilog—Unified Hardware Design, Specification, and Verification Language*, IEEE Std 1800-2009.

[54] *IEEE Standard for SystemVerilog—Unified Hardware Design, Specification, and Verification Language*, IEEE Std 1800-2012.

[55] John Cooley, "VHDL, the new Latin," *EETimes*, April 7, 2003. [Online]. Available: http://www.eetimes.com/document.asp?doc_id=1216865

[56] Richard Goering, "Designers debate 'VHDL is dead' assertion," *EETimes*, April 25, 2003. [Online]. Available: http://www.eetimes.com/document.asp?doc_id=1216820

[57] Michael Santarini, "Synopsys executive predicts end of VHDL," *EETimes*, April 11, 2003. [Online]. Available: http://www.eetimes.com/document.asp?doc_id=1216860

[58] *IEEE Standard VHDL Language Reference Manual*, IEEE Std 1076, 2000 Edition.

[59] *IEEE Standard VHDL Language Reference Manual*, IEEE Std 1076-2002.

[60] *IEEE Standard VHDL Language Reference Manual*, IEEE Std 1076-2008.

[61] Peter J. Ashenden and Jim Lewis. *VHDL-2008: Just the New Stuff*. Morgan Kaufmann, 2007.

[62] Jim Lewis, "VHDL-2008, The End of Verbosity," MAPLD 2013. [Online]. Available: http://www.synthworks.com/papers/VHDL_2008_end_of_verbosity_2013.pdf

[63] Jim Lewis, "VHDL's OSVVM, The Death of SystemVerilog?" MAPLD 2013. [Online]. Available: http://www.synthworks.com/papers/VHDL_OSVVM_the_death_of_SV_2013.pdf

[64] Cliff Cummings, "SystemVerilog – Is This The Merging of Verilog & VHDL?" SNUG Boston, 2003.

[65] Stu Sutherland and Don Mills, "Synthesizing SystemVerilog: Busting the Myth that SystemVerilog is only for Verification," SNUG Silicon Valley, 2013.

[66] Victor Berman, "An Analysis of the VITAL Initiative," Fall 1993 VIUF.

[67] Synopsys SolvNet documentation, January 2016.

[68] Cadence online documentation, latest version, January 2016.

[69] Mentor Graphics applications engineer, private communication.

[70] ATopTech online documentation, latest version, January 2016.

[71] Stuart Sutherland, "Is SystemVerilog Useful for FPGA Design? ('Burn and Learn' versus 'Learn and Burn')," SNUG San Jose, 2009.

[72] Mike Baird, Willamette HDL, private communication.

[73] Doug Smith, "Using `bind` for Class-based Testbench Reuse with Mixed-Language Designs," SNUG San Jose 2009.

[74] John Aynsley. "VHDL versus SystemVerilog," Doulos, 2012. [Online]. Available: https://www.youtube.com/watch?v=mqWMHQ7QwQU#t=496 and https://www.doulos.com/knowhow/video_gallery/#anchor01

[75] Harry Foster, "Part 8: The 2010 Wilson Research Group Functional Verification Study," Mentor Graphics, May 13, 2011. [Online]. Available: https://blogs.mentor.com/verificationhorizons/blog/2011/05/13/part-8-the-2010-wilson-research-group-functional-verification-study/

[76] Synopsys Formality R&D team, private communication, March 2015.

[77] Shannon Hilbert, "Verilog vs. VHDL," Feb. 4, 2013. [Online]. Available: http://www.bitweenie.com/listings/verilog-vs-vhdl/.

[78] Howard A. Landman, private communication.

[79] Dennis Brophy, "The IEEE's Most Popular EDA Standards," Mentor Graphics, June 17, 2011. [Online]. Available: https://blogs.mentor.com/verificationhorizons/blog/2011/06/17/the-ieees-most-popular-eda-standards/

[80] John Cooley, "Another Religious War Revisited," SNUG 00 Item 7, 5 April 2000. [Online]. Available: http://www.deepchip.com/items/snug00-07.html

[81] Ronald Collett, "VHDL by TKO," *Electronic Engineering Times*, p. 42, January 4, 1993.