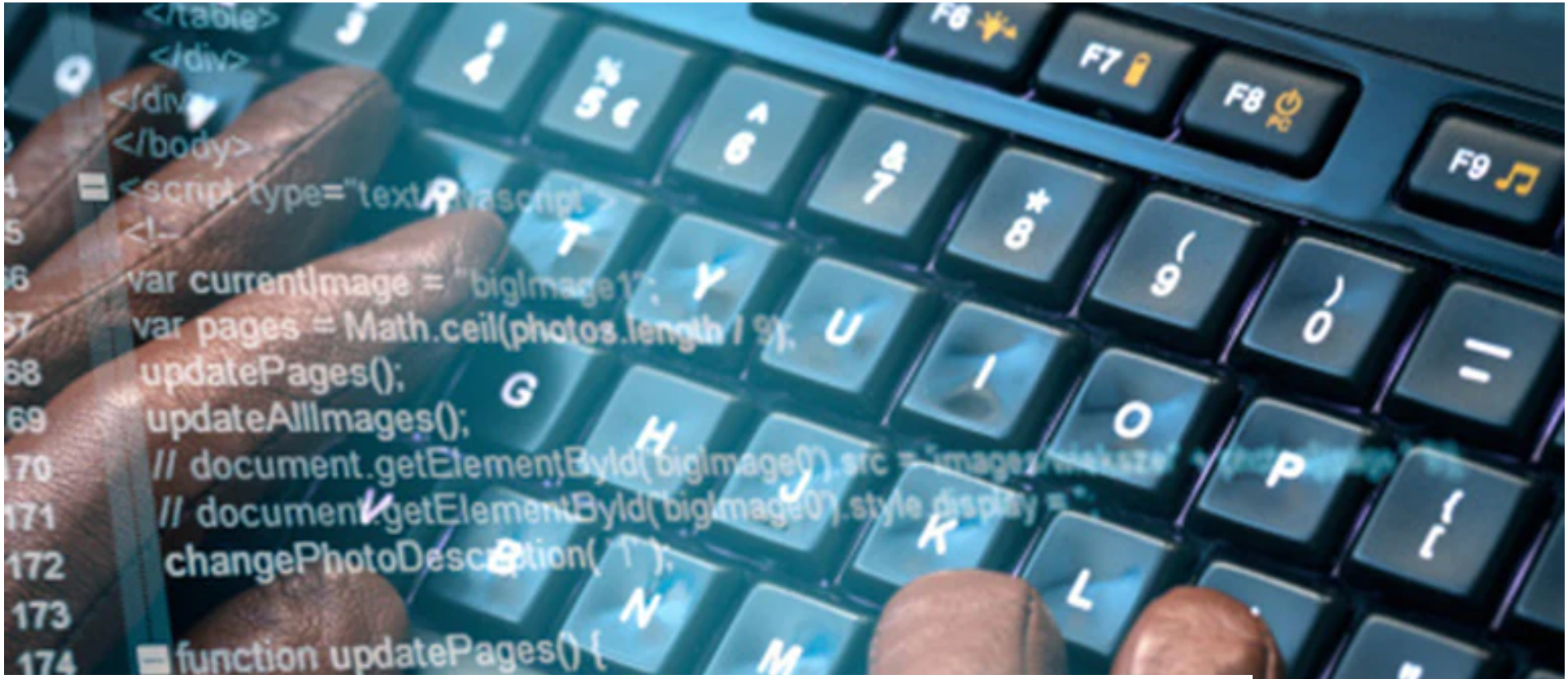

What's the Difference Between VHDL, Verilog, and SystemVerilog?

Rob Dekker

SEP 17, 2014

Rob Dekker is CTO and Founder of Verific Design Automation.

Designers of electronic hardware describe the behavior and structure of system and circuit designs using hardware description languages (HDLs)—specialized programming languages commonly known as VHDL, Verilog, and SystemVerilog. They differ from software programming languages because they include a means of describing propagation time and signal strengths. These days, it would be impossible to design a complex system on a chip (SoC) for a mobile device or any other consumer electronics product without an HDL.

Each of the three HDLs has its own distinct style. VHDL and Verilog implement register-transfer-level (RTL) abstractions. When they were first introduced in the late 1980s, they were considered breakthrough technologies

SystemVerilog was developed to provide an evolutionary path from VHDL and Verilog to support the complexities of SoC designs. It's a bit of a hybrid—the language combines HDLs and a hardware verification language using extensions to Verilog, plus it takes an object-oriented programming approach. SystemVerilog includes capabilities for testbench development and assertion-based formal verification.

The U.S. Department of Defense funded VHDL, and Gateway Design Automation developed Verilog to drive the Verilog simulator. Cadence Design Systems, once it acquired Gateway, placed the Verilog HDL into the public domain and it became an industry standard.

In the early 1990s, the electronics industry was forced to contend with the "language wars," where competing factions in either the VHDL or Verilog camp competed for an engineer's mindset and desktop. Both languages survived and now coexist, often in the same design flow.

Co-Design designed SystemVerilog, initially known as SUPERLOG. After Synopsys acquired Co-Design, SUPERLOG/SystemVerilog was placed in the public domain as well.

All three are IEEE industry standards –– VHDL is IEEE 1076-2008, Verilog is IEEE 1364-2005 and SystemVerilog is IEEE 1800-2012. The IEEE works cooperatively with the industry standards organization

My company, Verific Design Automation, has built parsers and elaborators for VHDL, Verilog, and SystemVerilog since 1999.

Download this article in .PDF format
This file type includes high resolution graphics and schematics when applicable.

Comparing VHDL, Verilog, SystemVerilog

VHDL and Verilog are considered general-purpose digital design languages, while SystemVerilog represents an enhanced version of Verilog. Each has its own style and characteristics. VHDL has roots in the Ada programming language in both concept and syntax, while Verilog's roots can be tracked back to an early HDL called Hilo and the C programming language.

With Verilog and VHDL, engineers can represent the desired functionality as a software program. Then the model is simulated to confirm the design will work as intended. Any problems can be corrected in the model, and simulation will verify the correction.

to do extra coding to convert from one data type to another. VHDL often catches errors missed by Verilog. VHDL emphasizes unambiguous semantics and allows portability between tools.

Sample VHDL Code

```
reg1: process (rst, clk)
    begin
    if  rst = '1' then
        q_reg <= (others => '0');
        q_i <= (others => '0');
    elsif  rising_edge(clk) then
        if  s_l = '1' then
        q_i(0) <= q_i(7);
        loop1:  for i in 6 downto 0 loop
            q_i(i + 1) <= q_i(i);
        end loop loop1;
        q_reg <= y;
        else
        q_i <= q_reg;
```

```
    end if;
    end process reg1;
```

Verilog is weakly typed and more concise with efficient notation. It is deterministic. All data types are predefined in Verilog and each has a bit-level representation. Syntax is C-like.

Sample Verilog Code

```
always @(posedge CLK or posedge RST)
begin
         if (RST) begin
                q_reg = 0;
        Q = 0;
        end else if (S_L) begin
                Q[7:0] = {Q[6:0],Q[7]};
 q_reg = Y;
        end else begin
                Q = q_reg;
        q_reg = Y;
```

SystemVerilog includes a set of extensions to the Verilog HDL to help engineers design and verify larger and more complex designs. In fact, many industry watchers consider it the first Hardware Description and Verification Language (HDVL), because it combines VHDL and Verilog features with those of Hardware Verification Languages (HVLs) Vera and *e*, as well as C and C++. It's targeted at RTL coding, using constrained random techniques for assertion-based and coverage-driven verification.

Sample SystemVerilog Code

```
property p_push_error;
    @ (posedge clk)
   not (b_if.push && b_if.full && !b_if.pop);
  endproperty : p_push_error
  ap_push_error_1 : assert property (p_push_error);
  property p_pop_error;
   @ (posedge clk)
   not (b_if.pop && b_if.empty);
  endproperty : p_pop_error
  ap_pop_error_1 : assert property (p_pop_error);
```

Because of its structure, VDHL catches most errors early in the design process. Verilog, on the other hand, enables engineers to quickly write models. SystemVerilog attempts to capture the best features of both, and includes features of HVLs to support testbench development and formal verification techniques.

Conclusion

The "language war" days are long over, as engineering teams worldwide effectively employ VHDL, Verilog and SystemVerilog for their SoC design needs. This overview provides a glimpse at the differences between the three. In general, SystemVerilog can help with both design and verification. Design tools from electronic-design-automation (EDA) companies can combine VHDL and Verilog to suit all types of engineering requirements.

In 1999, Rob Dekker formed Verific, a provider of hardware-description-language (HDL) source code software. Today, Dekker and a team of dedicated engineers develop parsers and elaborators for SystemVerilog, Verilog, and VHDL that have been used as the front-end software for synthesis, simulation, formal verification, emulation, debugging, virtual prototyping, and design-for-test applications.

Prior to founding Verific, Dekker was a software developer, manager, and director at Exemplar Logic, now part of Mentor Graphics. He was the architect and a primary developer of Leonardo, synthesis software used by field-programmable-gate-array (FPGA) designers. Dekker started his career with Philips Research in the

VOICE YOUR OPINION!

This site requires you to login or register to post a comment.

Latest Comments

Posted by donald.mccallum Nov 7th, 2016 7:39pm

While it was not possible to create what is now called a system on a chip with early FPGAs, it was quite possible to create quite sophisticated subsystems on a chip mated to external processors even in the days of schematic based design. I'll grant you that I would not want to try to fill a Virtex 7 full of logic using schematics today, but it would still be possible if not efficacious. Even before the FPGA companies started putting hard processor macros in their chips, we were putting soft processor macros in later chips creating the first (programmable logic based) SOCs.By that time I had switched to VHDL. The one thing I regret is that in the industry's race to put ever more functionality into ever more capable chips some of them have forgotten the fact that there are still many applications for half-pint solutions that an FPGA or CPLD can do quite readily. For these systems, RTL methods still work quite well. As an example, in the elevator business, we are not allowed to put "all of our eggs in one basket", typically using 3 uCs and an FPGA or CPLD to prevent a single point of failure from causing an elevator accident which as you might be able to visualize can be quite horrific, and to be avoided at all costs. All of which should illustrate why I do not depend on things like the C language to "abstract" my hardware (away to obscurity) so I don't have to think about it. I WANT to think about it. Long and hard. Under normal conditions and disaster conditions. But for your cell phones, pads, dpas and the like ad infinitum, by all means abstract away. The whole system goes down in a disaster anyway so who cares if the handheld digital device no longer works. With the possible exception of the GPS subsystem in the phone which SHOULD continue to work no matter what else is going on. Who knows, you might need to get home in a world of shattered landmarks. Become a duly licensed amateur radio operator if you think you might still want to communicate farther than you can shout under those same conditions.

Posted by donald.mccallum Nov 7th, 2016 6:55pm

critical application due to its nasty habit of allowing programmers to create faulty code and not flagging it, usually due to its weakly typed nature. Today the use of C in critical code in automotive and other vehicular applications is restricted to a subset of C and requires automated code checking to catch any logical traps that are still possible to create. In my opinion I prefer strongly typed languages such as VHDL and Pascal that never relinquish their forced correct code by design paradigm starting from the first line of code onward. It doesn't allow my lazy ways to create trash code. I like the self documenting feature also, but I usually spend a lot of extra time documenting code intent in sometimes rather lengthy soliloquies about what the code is supposed to do rather than a blow by blow account of how it does it. Finally, at some point the code, no matter how ingenious, must be reduced to something that can be downloaded into hardware and tested there with simulators that grok hardware. The idea that C source can be created which will produce optimized working hardware without getting your hands dirty in the hardware is something only a software engineer could believe. And I wouldn't bet my life and well being on it. My two cents worth.

Posted by roger.engelke Sep 19th, 2014 3:29pm

You are right about the wrong code in the VHDL spot. It was a mistake in posting the ariticle, and has now been fixed.--Ed

Posted by matthieu.wipliez Sep 19th, 2014 3:03pm

The first example "Sample VHDL Code" is Verilog code... which uses a bad coding style: no clear separation between what happens at reset and what happens on a clock edge, use of blocking assignments with registers. The second example "Verilog code" is SystemVerilog!

Yes "the 'language war' days are long over", and that is too bad. Verilog and VHDL are equally bad but for different reasons, and while SystemVerilog improves things for verification, it does not help much for design. The thing is that RTL remains a hopelessly low level of abstraction.

Take a look at the Cx language, and you will see that it is possible to design efficient hardware without the pain of RTL and HDLs :-)

IEEE's Two-fer Deal: Two Virtual Symposia on VLSI Technology & Circuits for One Fee

JUN 08, 2020

Analog

Find the Fastest Route to Portable Stimulus Tests with SystemVerilog

NOV 12, 2019

Test & Measurement

What's the Difference Between a BSP and SDK?

JUL 01, 2015

Dev Tools

Transaction-Based Emulation Helps Tame SoC Verification

SEP 03, 2014

EDA

MARKETS > AUTOMOTIVE

What's the Difference Between Bluetooth Low Energy, UWB, and NFC for Keyless Entry?

Automotive keyless entry brings convenience, but this feature gives hackers a new way to unlock or steal cars. Learn about the communication protocols involved and how UWB can make this more secure.

Joseph Sfeir

AUG 31, 2020

- How automotive keyless entry is giving hackers a new way to steal cars.
- The three communication protocols involved in automotive keyless entry.
- How UWB works to make automotive keyless entry more secure.

With the COVID-19 pandemic shelter in place and significant numbers of employees now working from home, consumers are driving less and, unfortunately, seeing an increase in auto theft. In fact, with cars left unattended for longer periods of time, the Associated Press reported in May 2020 that major cities like New York City and Los Angeles have seen an increase in auto theft.

Even before this "new normal" we're experiencing in the age of COVID-19, the National Insurance Crime Bureau (NICB) reported that 209 vehicles, on average, were stolen *each day* across the U.S. due to drivers forgetting key fobs in their vehicles, making for easy theft.

While leaving key fobs behind is an unfortunate mistake leaving drivers vulnerable to theft, another vulnerability is making car thefts easier—smart keys and smartphone car access.

While keyless entry is a standard on many cars sold today, the feature isn't totally secure. For all of the convenience smart keys and smartphone access brings to drivers, it has given hackers a new way to unlock or

Three communication protocols are involved in enabling a smart key or smartphone for unlocking a vehicle: Bluetooth Low Energy, ultra-wideband (UWB), and near-field communication (NFC), the latter mostly used as a backup. The three communication protocols, however, aren't equal in terms of access security.

Bluetooth Low Energy

For quite some time, Bluetooth Low Eenergy (BLE) has been used in vehicles to unlock/lock the car and connect multimedia applications—to pair a smartphone with the multimedia console for voice calls or music streaming applications.

BLE is one of the communication technologies used for smart key or smartphone wireless access when approaching a car. In newer car keys, BLE communication is mainly used to track the approach of a driver beyond the 10-meter distance, while also preparing for UWB authentication. By using data packets, BLE relies on the measurement of signal strength to evaluate the distance of a driver.

Unfortunately, despite a certain degree of encryption, BLE can be subjected to jamming and relay or man-in-the-middle attacks. During a relay attack, the communication from the valid key is spoofed by a hacker by amplifying its signal strength and tricking the receiver into believing that the key is nearby. If a hacker can sniff and replay the data exchange between key and car, it's possible to unlock the car and steal it.

Ultra-wideband, based on IEEE802.15.4z, is a newer technology standard that can be employed in wireless entry systems to prevent distance manipulation attacks—short UWB pulses are used for precise and secure time-of-flight (ToF) and angle-of-arrival (AoA) measurements. ToF measures the propagation time that it takes for the RF signal to travel between the transmitter and receiver. AoA measures the angle of an incoming signal using multiple antennas. Positions can be determined by having multiple angles, multiple distances, or a combination of the two.

With UWB, once the two devices (in this case, the smart key/smartphone and car) is within proximity, they begin ranging and calculating the distance with a centimeter level of precision between them. The smart key/smartphone can lock or unlock the vehicle when it's within, for example, two meters of the vehicle, depending on the direction of movement (moving away or towards).

Unlike BLE, UWB is based on time, not signal strength. Therefore, a relay attack will not work on UWB, as the attack will add latency to the transmission and indicate that the key is actually far away from the receiver. In addition, adding a Scrambled Time Sequence (STS) into the UWB frame—an "encrypted" measure of a timestamp—prevents preamble insertion attacks and allows for even more accurate distance measurements. *Figure 1* shows BLE and UWB distance for automotive access.

1. Shown are the Bluetooth Low Energy and UWB distances for automotive access.

Redundancy with NFC

NFC provides the same locking and unlocking capability as UWB or BLE, but NFC is mainly used as a redundant system in the event the smart key's or smartphone's battery runs out. Redundancy in automotive applications is very important because a user doesn't want to be in a position of not being able to unlock the car.

When an NFC passive device is brought within the near field of an active device (the car), the passive device "wakes" and communicates with the active device to perform an action, like unlocking the door.

While NFC is a much simpler communication protocol and therefore doesn't have the same security benefits as UWB, it's an excellent backup system for UWB because it's very low power and requires much less battery from the device being used. In some cases, the smart key or smartphone may have a capability built in to recognize the level of power supply that's available at a specific moment and select which communication protocol should be used to unlock the vehicle. It is, however, a less convenient solution than UWB because the car key needs to be held on an active part of the car.

Conclusion

2. The chart compares the important technical aspects of each communication protocol.

While a UWB chip is more expensive than BLE, the wireless protocol provides significantly greater security in ensuring that only the driver obtains automotive access. Successful deployments of UWB-enabled smart keys or smartphones will depend on the accuracy of their fine ranging capabilities, making test compliance verification and performance validation imperative to successful implementation.

Joseph Sfeir is Director of Business Development at LitePoint.

Wireless Race

OCT 20, 2020

Industrial Automation

AUG 10, 2020

Industrial Automation

New Applications Bring Renewed Life to UWB Wireless

APR 01, 2019

Industrial Automation

What's The Difference Between Measuring Location By UWB, Wi-Fi, and Bluetooth?

FEB 06, 2015

Communications

Load More Content