DIGITAL INDUSTRIES SOFTWARE

The rise and fall of synthesis bugs in safety-critical FPGAs

IEC 61508 / ISO 26262 / EN 50128 / DO-254

Executive summary

FPGAs are the dominant hardware platform in low-volume, safety-critical applications, including aerospace and nuclear power plants. Modern FPGAs allow for the implementation of high-performance designs with integrated safety mechanisms. This is driving adoption in additional industries, including automotive.

Functional safety standards require a rigorous development process to minimize the risk of introducing systematic faults. Some RTL issues may only reveal themselves as bugs in the post-synthesis netlist. Additionally, synthesis tools manipulate the design to map it into the fixed FPGA structure. These complex transformations present a high risk of introducing bugs.

Gate-level simulation and lab testing can only cover a tiny portion of the FPGA functionality and are likely to miss implementation bugs. Moreover, they are slow to run and hard to debug.

This paper presents an implementation signoff flow proving that the final FPGA netlist is functionally equivalent to the RTL model. Based on FPGA-specific, mature formal technology, the solution is exhaustive and efficient, with many issues being caught before synthesis starts.

Sergio Marchese
Siemens EDA

Contents

Safety-critical FPGAs

Field-programmable gate arrays (FPGAs) are the dominant hardware platform in many safety- critical, low-volume applications, including aerospace and nuclear power plants (NPPs). Modern FPGA devices feature integrated microprocessor cores, digital signal processing (DSP) units, memory blocks and other specialized intellectual properties (IPs). These advanced devices allow for the implementation of large, high-performance system-on-chip (SoC) designs with integrated safety mechanisms, making a strong case for adoption in additional safety-critical applications traditionally dominated by application-specific integrated circuits (ASICs). A notable example is automotive hardware for AI, which, driven by the challenges of autonomous vehicles, must serve the demands of an expanding range of applications, including sensor data fusion and processing.

At a high-level, the FPGA and ASIC development flows are similar. Register- transfer level (RTL) coding and integration of third-party intellectual properties (IPs) are crucial steps in the front-end part of the flow. Extensive functional verification of the RTL design model reduces the risk of mismatches between requirements and RTL behavior. At this stage, specification, coding, and module integration mistakes are the main source of systematic faults that, if undetected, could lead to dangerous failures of the FPGA device in the field.

The RTL model then goes through several implementation steps (see figure 1). Synthesis and place-and-route tools map the design onto the target FPGA device. The bitstream generation step produces the file used to program the FPGA. Functional verification of implementation steps, the focus of this paper, reduces the risk of mismatches between the derived netlists and the RTL design model. This is crucial to close the loop and ensure that the functionality implemented in the FPGA device matches the hardware requirements.

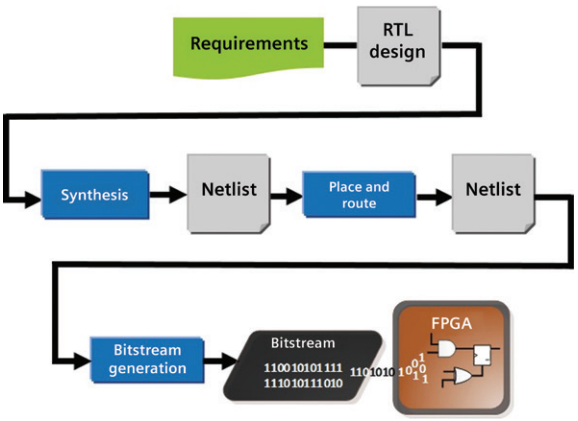

Figure 1. The FPGA development flow.

The 2016 Wilson Research Group Functional Verification Study (see figure 2) shows that for a significant number of FPGA projects one or more bugs escape into production devices. The same study also reports that 78% of safety-critical FPGA designs have bug escapes into production.

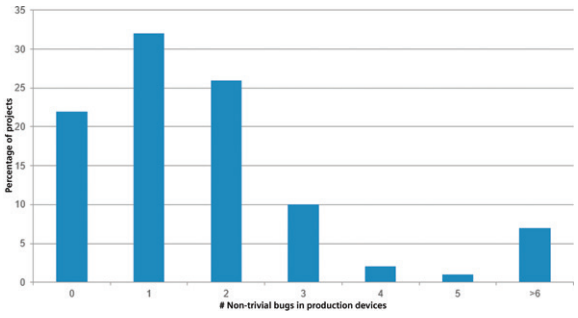

Figure 2. Non-trivial bug escapes in FPGA production devices.

FPGA implementation bugs reported on industrial projects include incorrect FSM re-encoding, wrong bit ordering in bus connections, incorrect settings for RAM parameters, routing issues, and introduction of additional, unspecified logic as well as retiming.

Besides the cost advantages of FPGAs over ASICs for low-volume applications, another crucial benefit of FPGAs is that if, for whatever reason, the final hardware function needs to be changed, the whole flow can be repeated, and a new bitstream generated to re-program the same device without the large non-recurring engineering (NRE) and fabrication delay costs of ASICs. However, shipping FPGAs with functional bugs is not acceptable, particularly in safety-critical applications.

This paper presents an efficient, rigorous verification signoff flow that detects functional bugs introduced during FPGA implementation.

Implementation issues in FPGA flows

Functional bugs can be introduced during implementation steps, either because of RTL issues that cannot be detected during synthesis, or because of malfunctions in implementation tools, particularly synthesis and place-and-route, corrupting the original RTL functionality. While engineers may expect that implementation tools have been extensively tested prior to release, each design and coding style is unique and may trigger unknown corner cases.

Simulation-synthesis mismatches

Regardless of the implementation tool used, there are coding issues that may go undetected during RTL verification and creep into the synthesis netlist. As an example, consider a Verilog array indexed with a signal that may take values outside the bounds of the array (see figure 3). Indexing an array signal outside of its bounds creates an "X" during simulation. In some corner case scenarios, RTL simulation behavior may not match the behavior of the corresponding netlist in the presence of unknown, or X, values. Consequently, while the synthesis tool operates correctly, its generated netlist may still not match the intended RTL behavior.

```
reg [2:0] i;
reg [5:0] array;

always @(posedge clk)
for(i=0 ; i<=M; i=i+1)
  array[i] <= 1;
```

Figure 3. Potential array indexing issue.

Requirement-based testing does not explicitly target these type of corner cases, which can therefore be missed. RTL linting tools may warn about these scenarios. However, they provide neither a definite answer nor a simulation trace that shows how the bad scenarios may occur. While valuable, linting tools give little help with debug and, particularly for these types of issues, raise many false alarms and tend to be noisy.

Implementation optimizations

Unlike in ASIC, FPGA implementation tools do not have a white silicon canvas on which to "draw" the RTL design. Instead, they must fit the desired functionality into a prefabricated structure while meeting performance and power consumption goals. Implementation tools perform significant changes to the original logic structure of the design to improve device utilization and overall quality of results (QoR). Design transformations range from simple renaming of registers to complex retiming (see figure 4). Advanced optimizations pose a higher risk of corrupting the RTL functionality.

Configurable logic blocks (CLBs) contain registers and lookup tables (LUTs) and are fundamental building blocks in the FPGA fabric. The CLB elements are connected through a programmable switch matrix. The logical function implemented by a LUT depends on how it is instantiated and parametrized within the synthesis netlist. Incorrected parametrization of a LUT corrupts the intended logical function.

FPGAs also contain distributed and block random-access memories (RAMs). Synthesis tools use RAMs to optimize the data path and improve timing. Even for a simple RTL array, advanced algorithms are employed to identify the best choice of memory elements. Moreover, large memories may need to be mapped into smaller FPGA RAMs, which individually do not have the word width or addressability required by the RTL design. This results in complicated correspondences between the RTL and the synthesis netlist.

The encoding of FSMs states in the RTL design may also be transformed during synthesis to optimize power, timing, and device utilization. Synthesis tools feature multiple re-encoding algorithm that may perform optimizations that cross hierarchies and even alter the overall number of states in the FSM representation.

Certain design styles of shift registers (SR) may result in the conversion of the SR chain to a memory block with intelligently modelled control logic to produce an equivalent behavior. Successive shifting of the input data in the RTL registers is transformed to write operations to distinct memory locations in the netlist. While this transformation smartly utilizes the available FPGA resources, it also places a great deal of responsibility on the synthesis tool.

Retiming is another commonly employed optimization technique. Its main goal is to improve timing and increase the clock frequency at which the FPGA

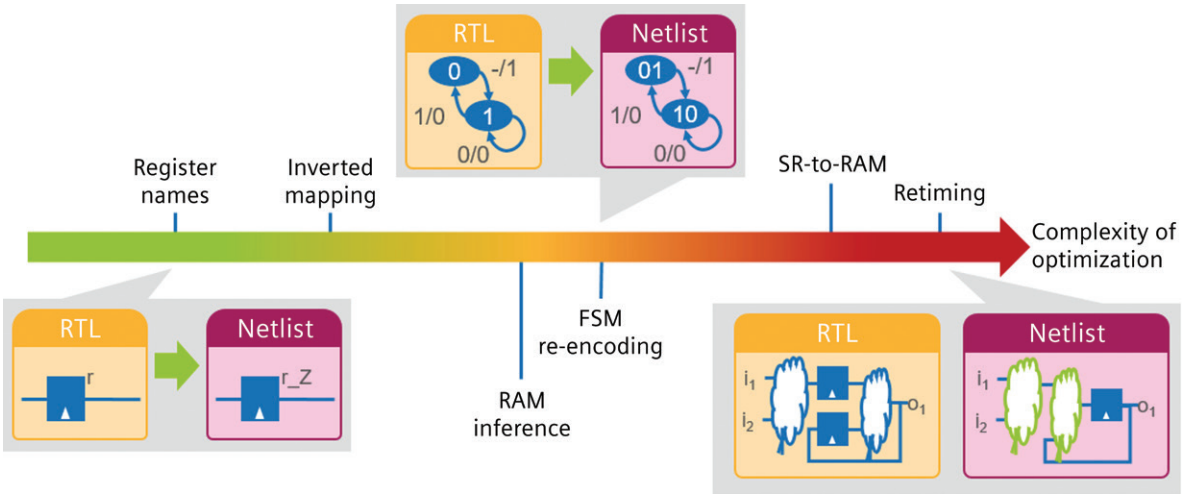

Figure 4. Complexity of FPGA implementation optimizations.

may operate. As logic functions are moved across register stages, retiming is also a high-risk design transformation.

Depending on the target FPGA family and implementation flow, design optimizations may be performed during synthesis, initial place-and-route or even after initial routing, for example to enable retiming algorithms to leverage more accurate estimates of connection delays.

Insertion of safety mechanisms

Certain FPGA synthesis tools support the automatic insertion of hardware safety mechanisms. Safety mechanisms do not change the design functionality when no fault is present. In the event of a random hardware fault occurring during field operation, they need to raise an alarm and potentially correct the effects of the fault on the fly.

Synthesis tools may transform the encoding of FSMs, for example to include Hamming-based error detection and correction. Triple modular redundancy (TMR) is another type of safety mechanism where a critical logic function is triplicated and voting logic added to determine which of the three outputs should be considered as correct. Logic duplication and inference of memories with error correcting codes (ECC) may also be supported. Users certainly benefit significantly from these design enhancements that can be performed automatically by the synthesis tool. However, the inserted logic could contain bugs and must be rigorously verified.

Traditional verification of implementation steps

Extensive gate-level simulation (GLS) and lab testing may detect functional bugs introduced during implementation steps. However, these techniques have significant shortcomings. An effective verification flow must detect bugs as soon as possible once they are introduced: the later a bug is found the higher its cost. Detecting an RTL issue or synthesis bug during lab testing or GLS of the place-and-route netlist is inefficient. Moreover, the effect of bugs introduced by implementation tools is unpredictable. Simulation tests are not intended to verify the correctness of the implementation tools, and even running all available tests at gate-level, which is in fact impractical in most cases, only provides limited confidence. This approach cannot be - and is indeed far from - exhaustive. Finally, debugging GLS and lab test failures is hard and time consuming.

To reduce the risk of introducing errors, some engineers switch off advanced synthesis optimizations options and avoid using the latest tool versions. This approach is inefficient and inadequate for many modern designs. Moreover, it reveals an overall lack of confidence in the verification flow.

RTL coding issues and advanced optimizations during FPGA implementation increase the risk of introducing bugs. GLS and lab testing are late, slow, and miss corner-cases. Is there a better solution?

Formal signoff of FPGA implementation

Formal methods are widely recognized as a powerful verification technology. Unlike simulators, formal tools do not need input vectors and perform an exhaustive analysis of the design, covering all possible hardware behaviors and states. The use of formal tools is well established in ASIC development. Formal design inspection and exploration is valued for detecting both basic and corner-case RTL issues early and without the need for a simulation testbench. Moreover, driven by high NRE and fabrication delay costs, for nearly 20 years ASIC development teams have routinely applied formal combinational equivalence checking (EC) as a superior alternative to GLS to ensure that no functional bugs are introduced during synthesis, place-and-route, and engineering change orders (ECOs).

Inspection of RTL code

Automated, formal inspection of RTL code detects issues before synthesis starts. Unlike linting, formal tools provide a definite answer on whether an array may be indexed out of bounds. In this case, the tool provides an easy-to-debug simulation-like trace, or counterexample, that demonstrates how the design misbehaves.

DV-Inspect™ from Siemens EDA, a part of Siemens Digital Industries Software, performs numerous checks on the RTL design in addition to array out-of-bounds checks (see figure 5). The tool automatically creates assertions and verifies those assertions using formal proof engines. Many of the issues detected could result in simulation- synthesis mismatches or other issues that compromise the functionality of the synthesis netlist.

Automated formal RTL checks (prior to synthesis)		
Array/Range out of bound	SNPS full case	Negative/Zero division, exponent, reminder
Function without return	SNPS parallel case	X/Z signal/ variable resolution
X/Z state value	Write-write case	Arithmetic shifts

Figure 5. Automated formal RTL checks prior to synthesis.

DV-Inspect has been used in hundreds of large industry designs, including safety-critical FPGAs. Many DV-Inspect users had no previous experience with formal tools.

Equivalence checking for FPGAs

Formal EC tools can mathematically prove (or disprove) that two designs are functionally equivalent. This is the most rigorous way to ensure that synthesis and other implementation steps have not introduced bugs. The input design to the implementation tool is typically named golden design. The generated netlist is named revised design.

Combinational EC largely relies on one-to-one mapping of states between golden (e.g., the RTL) and revised (e.g., post-synthesis netlist) designs. Through state mapping, the complex problem of proving that two large designs are functionally equivalent can be split into a multitude of much simpler problems: comparing the functionality of two combinational logic cones (see figure 6).

The design transformations performed by FPGA implementation tools significantly break one- to-one state mapping. Formal sequential EC algorithms can

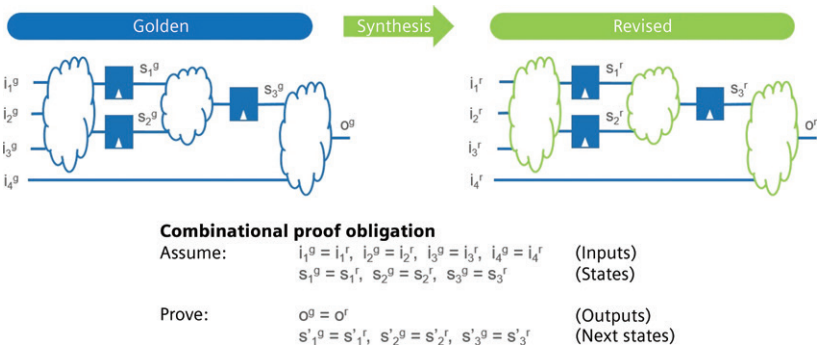

Figure 6. Proof obligation in combinational equivalence checking.

prove equivalence of sequential logic cones, thus not requiring state mapping. However, while these algorithms have improved dramatically in recent years, they do not scale. Partial state mapping is necessary to leverage combinational EC wherever possible and apply sequential algorithms only on limited design portions (see figure 7). In this context, identifying corresponding states is a crucial, challenging task. Manual mapping is tedious and time-consuming. Mistakes waste engineering resources.

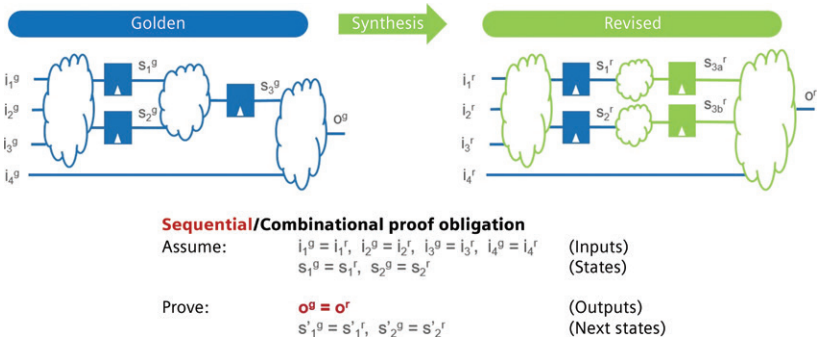

Figure 7. Proof obligation in sequential and combinational equivalence checking.

EC-FPGA™ performs automated, accurate mapping through a variety of techniques, including name-based mapping algorithms, hint-based algorithms leveraging information provided by synthesis tools, for example on state re-encoding, and algorithms that leverage known behavior of implementation tools for the specific FPGA target device. Advanced design transformations, including FSM re-encoding and retiming, are made more amenable to formal EC through specialized mapping algorithms that identify high-level relations between sets of golden and revised states. Finally, FPGA-specific sequential proof engines cover the areas where no mapping exists or can be found.

Thanks to these advanced algorithms, Siemens EDA brings the power of automated formal EC to FPGA designs. To achieve further device-specific automation and high performances, Siemens leverages close partnerships with leading FPGA vendors: Xilinx, Intel, and Microsemi. However, EC-FPGA works independently from implementation tools. The use of hint files and other similar approaches is conservative in the sense that the information is used but not trusted to be correct.

Verification of safety mechanisms

Safety mechanism inserted by synthesis tools must be rigorously verified. EC- FPGA automatically handles the crucial no-fault scenario, and exhaustively proves that the additional logic does not change the RTL functionality when no random hardware fault is present. In addition, engineers must verify that the safety mechanism functions as expected when faults occur. The Siemens EDA platform supports fault injection and verification through a portfolio of safety analysis and verification apps. The level of verification automation depends on the type of safety mechanism. For a logic duplication mechanism, for example, the injection of faults and verification of the output's comparator logic can be fully automated through the Fault Detection Analysis (FDA™) App.

Formal signoff flow

Formal verification signoff (see figure 8) enables engineers to use advanced FPGA optimizations and the latest synthesis technology with confidence. Formal RTL design inspection is more powerful than linting and finds issues early, prior to synthesis. Formal EC proves that the golden design functionality is not corrupted by the implementation step. Finally, formal fault injection and verification, supported by specialized safety apps, can automate the verification of safety mechanisms in the scenarios when faults occur.

With this flow, weeks of GLS and lab testing can be replaced with hours of formal tool runtime. Siemens EDA tools share a common platform, do not need a testbench, and may take as little as a few hours to set up. Debugging is also much faster as issues are detected early in the flow, the portion of the design containing the bug is closely identified, and a simulation-like trace is provided.

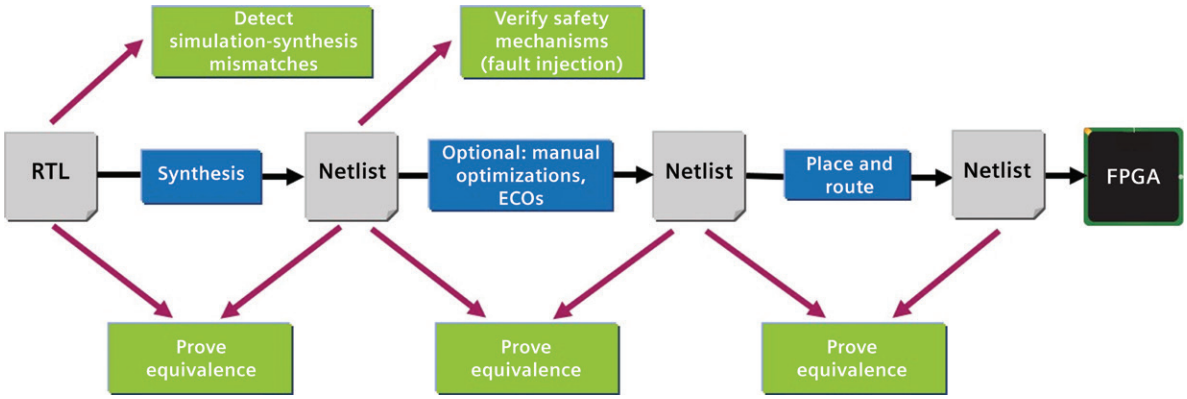

Figure 8. Functional signoff of FPGA implementation with formal verification.

Compliance with functional safety standards

Functional safety standards such as DO-254 for aerospace applications, IEC 61508, and its derivatives, including ISO 26262 for automotive applications, require rigorous hardware development processes that minimize the risk of systematic faults. Engineers must also demonstrate an adequate level of confidence in the use of software tools that tailor or replace activities required by the standard. Although verification tools may not introduce errors in the design flow, they may fail to detect errors and are also covered by these requirements.

Tool safety compliance is a project-specific task. However, vendors can provide safety certificates and standard-specific qualification kits that take most of the burden of safety compliance off the shoulders of users. In this context, safety compliance support is an integral aspect of state-of–the-art formal verification tools.

Siemens TÜV SÜD certification

EC-FPGA has tool qualification kits (TQKs) that enable users to meet the tool safety requirements of ISO 26262, IEC 61508 and EN 50128, without additional qualification effort, for up to ASIL D (ISO 262626 TCL2 and TCL3 tools) and SIL3 (IEC 61508 and EN 50128 T2 off-line tools) applications.

Siemens EDA engaged with TÜV SÜD, an accredited global testing, inspection, and certification provider, to perform a thorough, independent assessment of its organization, tool development and testing processes, as well as an inspection of its development facilities. The TÜV SÜD safety certificate and report form the basis of Siemens EDA's TQKs. Each TQK also includes tool documentation, a safety manual, and a system to track defects and notify users of newly found tool issues that might impact safety.

Siemens EDA DO-254 tool qualification kit

DO-254 provides guidance with a requirements-based process-oriented approach that significantly increases project lifecycle compared to a non-DO-254 compliant flow. The standard requires the assessment and, if necessary, qualification of software tools. Design tools, for example for synthesis, can introduce errors in the hardware. Verification tools, on the other hand, can fail to detect errors. The standard demands are more stringent for design tools.

The Siemens EDA tools used in the FPGA implementation functional signoff solution presented in this paper are verification tools. Depending on the specific use of the tools and project, compliance can be achieved through independent assessment of the tool outputs or through basic tool qualification. Siemens EDA provides a customizable DO-254 TQK and project-specific support to enable low-effort compliance with DO-254 tool qualification requirements.

Conclusion

FPGAs have long been the hardware platform of choice in many low-volume safety-critical applications. Nowadays, these devices can implement complex functions while fulfilling tough performance and power goals, thus competing with ASICs on high-volume safety-critical applications, including automotive.

The availability of advanced EDA tools and methodology is crucial to support this trend. ASIC development has used formal EC for nearly 20 years. Automated formal checks prior to synthesis are also widely adopted by ASIC teams. The same technology is now available in FPGA development, enabling a robust, efficient implementation process. The Siemens EDA formal signoff flow of FPGA implementation is orders of magnitude more rigorous and efficient than GLS and lab tests. The technology is mature and proven on hundreds of industrial designs for communications, NPPs, and other safety-critical applications. The Siemens EDA tools also come with TQKs that enable engineers to satisfy the tool safety compliance requirements of various functional safety standards, including IEC 61508, ISO 26262, EN 50128, and DO-254.

References and further reading

1. ISO 26262 Standard – Road vehicles – Functional safety, 2011.
2. IEC 61508 International Standard – Functional safety of electrical/electronic/programmable electronic safety-related systems, 2010.
3. RTCA/DO-254 Design assurance guidance for airborne electronic hardware, 2000.
4. DOT/FAA/TC-12/21 Qualification of tools for airborne electronic hardware, 2014.
5. Sergio Marchese. Using formal to verify safety critical hardware for ISO 26262, 2017.
6. Sergio Marchese. When correct is not enough – Formal verification of fault-tolerant hardware, 2017.
7. J. Grosse, S. Marchese. Shifting the burden of tool safety compliance from users to vendors, 2018.

Siemens Digital Industries Software

Americas: 1 800 498 5351

EMEA: 00 800 70002222

Asia-Pacific: 001 800 03061910

For additional numbers, click here.

About Siemens Digital Industries Software

Siemens Digital Industries Software is driving transformation to enable a digital enterprise where engineering, manufacturing and electronics design meet tomorrow. Xcelerator, the comprehensive and integrated portfolio of software and services from Siemens Digital Industries Software, helps companies of all sizes create and leverage a comprehensive digital twin that provides organizations with new insights, opportunities and levels of automation to drive innovation. For more information on Siemens Digital Industries Software products and services, visit siemens.com/software or follow us on LinkedIn, Twitter, Facebook and Instagram. Siemens Digital Industries Software – Where today meets tomorrow.

About the author

Sergio Marchese is the Technical Marketing Manager at Siemens EDA. He brings to this role 18 years of experience in the semiconductor and electronic design automation (EDA) industries. Marchese holds a Master of Electronic Engineering degree from University of Catania, Italy. He has presented at several industry conferences in Europe and North America and has published numerous papers.

siemens.com/software

84266-D2 2/22 H