embeddedworld2023
Exhibition&Conference
...it's a smarter world

Meeting Functional Safety Standards on Algorithm Implementation for FPGA and ASIC in a Dynamic Automotive Environment

Dimitri Hamidi
Application Engineering
The MathWorks GmbH
Munich, Germany
dhamidi@mathworks.com

Dr. Tjorben Gross
Application Engineering
The MathWorks GmbH
Munich, Germany
tgross@mathworks.com

Eric Cigan
Product Manager
MathWorks Inc.
Natick, USA
ecigan@mathworks.com

Tom Richter
Application Engineering
The MathWorks GmbH
Munich, Germany
trichte@mathworks.com

***Abstract*— FPGAs and ASICs are playing a greater role across an increasing variety of automotive systems and applications because of their high throughput, low latency, and superior per-watt performance. However, we see different challenges related to their development in the automotive industry: There is a need for efficient inter-team communication and collaboration in multidisciplinary FPGA/programmable SoC and ASIC projects. Changing requirements and shrinking design cycles require the ability to react faster. Standards, such as ISO 26262 for functional safety, must be fulfilled, while shorter project deadlines must be met. Finally, the global chip shortage drives the need for hardware-independent portable workflows, enabling more rapid adaptation to changes in the supply chain and smoother transitions from FPGAs to ASICs. This work explores an integrated workflow for designing and implementing signal processing, control design, and vision algorithms on FPGAs, programmable SoCs, and ASICs to address these challenges. We briefly cover the process-spanning requirements of authoring, architectural modeling, and modeling for HDL implementation, with verification and validation at each step. Furthermore, we cover related hardware/software codesign aspects. We show how an integrated, hardware-independent, and prequalified toolchain enables users to streamline ISO 26262 certification.**

Keywords—FPGA, SoC, ASIC, ISO26262, Functional Safety

I. INTRODUCTION

The electrical and electronics (E/E) components market is projected to grow at a 5.6% compound annual growth rate (CAGR). Autonomous driving and electrification of the powertrain are among the major drivers behind this trend [1]. Regarding ADAS/AD sensor systems, Radar and Lidar are expected to grow by 19.5% [2] and 13.5% [3], respectively. As the automotive industry climbs the ladder of autonomy levels, sensor resolution increases exponentially, requiring an enormous amount of (pre-) processing to convert the raw data into point clouds and detected objects. The complexity of signal processing algorithms is also growing to meet performance requirements. The need for higher throughput, low latency, and improved per-watt requirements often cannot be met with CPU or GPU implementations. In electrification, power electronics control is one major application. Power-efficient and low-vibration motor control requires complex algorithms with high sampling frequency and reaction times that often exceed what is achievable on today's CPUs [4].

As a result, FPGA and ASIC are increasingly the technology of choice, often used as application-specific accelerators on chips. At the same time, using embedded FPGAs (eFPGAs) is an emerging trend. FPGAs may be replaced with ASICs whenever justified by cost savings due to increases in production volume or the need for lower power requirements.

The growing complexity of FPGA and ASIC designs is challenging conventional development methodologies, leading to a disproportionate increase in required verification. SoC designs add another layer of verification complexity due to hardware/software interactions [5], especially in the presence of multi-core and multi-tasking subsystems.

The key question we address here is how to efficiently design and verify our algorithms for FPGA and ASIC in an automotive context, where device complexity keeps growing, design cycle time shrinks, and the market demands high quality and standard compliance. It is necessary first to understand the common challenges the industry is facing to answer this question.

A 2022 study by the Wilson Research Group (WRG) [5] indicates that more than two-thirds of all FPGA and ASIC projects are behind schedule. It also finds that more than half of the development effort on FPGA and ASIC projects is spent on verification activities, with debugging accounting for roughly a quarter of the project time. Despite these efforts, over 80% of all FPGA projects have non-trivial bug escapes into production, even in safety-critical applications. These bugs could be traced to logic and functional flaws in more than 50% of the cases. The study additionally finds that design errors are the leading cause of functional flaws, closely followed by problems related to changing incorrect and incomplete specifications. Table 1 summarizes some of the key findings. Overall, these numbers haven't changed significantly over the last decade. Development managers are using staff time to address the consequences of these issues instead of focusing on the next technical projects to help their companies grow and be more competitive.

We attribute these findings primarily to design and verification methodologies shortcomings based on translating textual requirements and behavioral specification models into RTL code and testbenches manually. They emerged over three decades ago and remain the predominant paradigm today, especially in safety-critical systems. In our further discussion, we will refer to this workflow as the traditional development process.

Next, we present an integrated workflow for digital hardware design and detail how it can address the challenges described above.

II. Model Based Design for Digital Hardware

Model-Based Design (MBD) is a mathematical and visual approach to developing complex systems that systematically use models throughout the development process. These models represent multi-domain, cyber-physical systems, including the environment, system components (e.g., electrical, AMS, RF, and mechanical), and software and hardware algorithms; these models allow designers first to understand behavior and find optimal design choices using simulation long before actual implementation. Engineers can generate optimized C/C++ and HDL code from models in the context of embedded hardware and reuse model simulation testbenches for deployment and verification.

Models serve as a common language throughout the development process and promote cross-functional team collaboration. Using models to refine product specifications reduces the dependency on prototype availability and is a key reason for faster product development and savings [6]. With MBD, an integrated toolchain is used from systems engineering throughout all project phases. It seamlessly connects algorithms to system architectures, hardware designs, and the verification process. MBD supports the core values of agile development [9].

A. System level specification

System requirements are created by separate teams and captured textually in a traditional development process using tools such as Microsoft® Word® or IBM® Doors®. At the same time, system architectures are specified in drawing tools, making them difficult to analyze, interpret, and manage as changes are made.

Validating these requirements can be difficult; an erroneous specification could result that would then be translated into design errors, as indicated in the WRG study. MBD begins with the same set of system requirements as a traditional process. However, it creates a system architecture with behavior and architecture models using MATLAB®, Simulink®, and System Composer™ instead of converting them into textual specification [7]. The textual requirements can be imported, managed, linked, and traced to the model's components as recommended by ISO 26262. This step helps to identify unintended functionality and ensures requirements coverage, and an executable specification of the system results. Engineers can simulate these architecture models to elucidate requirements and specifications, execute tradeoff analysis, and uncover inconsistencies and integration issues before implementation [6]. Known as Model Based System Engineering (MBSE), this approach can produce up to 55% overall savings after two years [8]. Using executable models reduces dependency on textual requirements and resolves ambiguities in product specification. MBSE is well in line with recommendations of ISO 26262 - Part 4, covering product development at the system level and allows to parallelize system and safety engineering [10].

B. Detailed algorithm design for implementation

Specifications are manually translated into HDL code in a traditional design process, which is time-consuming and error-prone. Defects may be introduced, accumulated, and propagated downstream at each phase. As a result, debugging consumes roughly one-quarter of the overall development effort [5]. The heterogeneous tool environment, multiple manual steps, changing requirements, and late-stage defect detection contribute significantly to the ubiquitous project delays [6].

MBD can alleviate those issues on multiple levels, and the key is automatic RTL code generation. HDL Coder™ [11] can generate VHDL or Verilog code from MATLAB®, Simulink®, and Stateflow®. All can be combined within Simulink, and the designer can choose a suitable data and control path design methodology. Simulink® is a graphical environment for hardware design, with models reflecting hardware architecture and hardware concepts such as parallelism. Multi-rate systems and asynchronous clock design are supported for code generation. The generated code can match the model behavior on the bit (numerical) and cycle level (timing), facilitating debugging and verification.

An extensive Simulink block library and MATLAB functions support automatic HDL code generation with proven efficiency [24][25][26] for different applications, including signal processing and communications, control design, and computer vision. Many blocks are configurable, offering micro-hardware architecture implementation choices. Fixed Point Designer™ [12] helps analyze and convert floating point algorithms to reduced-precision floating points or fixed points. Math functions can be automatically converted into interpolated lookup tables. HDL code generation supports floating points in IEEE 754 half, single, and double precision. HDL Coder™ supports resource sharing and retiming optimizations.

Moreover, resource mapping can be controlled for DSP slices, BRAM, and lookup tables. These capabilities enable design space exploration and evaluation in area, throughput, latency, and power tradeoffs. The code is customizable, readable, and well-structured, retaining model hierarchy and signal naming, with different comments, such as data types. IP cores with standard AXI interfaces can also be generated. Thus, production-quality RTL code can be achieved from models at high levels of abstraction.

Bidirectional traceability is supported as recommended by ISO 26262 because the HDL code is traceable to the model. RTL code is generated, and the model can be easily repurposed for different FPGA target devices or transitioned to ASICs since target-independent RTL code is generated. Manually written HDL code can be integrated using cosimulation with HDL simulators if required for system-level stimulation [13].

TÜV SÜD certifies HDL Coder™ as suitable for ASIL A-D and can be classified as tool confidence level 1 (TCL1), meaning that no additional qualification measures are required provided that a verification and validation (V&V) workflow (described below) is followed. The IEC Certification Kit [14] provides tool-qualification artifacts, certificates, and tool-validation test suites to streamline certification. This qualification lowers the certification burden and supports toolchain upgrades.

MBD supports hardware-software codesign for SoC platforms [15]. Algorithm models can be partitioned between implementation in programmable logic and processor cores. Production quality C/C++ code can be generated for the CPU using Embedded Coder®, enabling communication with IP cores via AXI4, AXI4-Lite, or AXI4-Stream interfaces. Embedded Coder is similarly qualified for ASIL A-D as TCL1, and the same model V&V tools can be leveraged, which we describe in the next section.

SoC Blockset™ [16] allows simulating internal AXI interfaces, DDR memory transactions, and multi-core task scheduling to further help with SoC designs. The user can design the data path and quickly explore hardware and software partitioning performance. The whole application can be deployed automatically for prototyping, and the simulated performance can be validated using on-device profiling.

Lastly, we assert that model-level debugging is more efficient than debugging RTL implementations. Model changes can be introduced rapidly. Simulink® library blocks are pre-tested, and models are transparent regarding data types and sizes and sampling times. Engineers can harness the data visualization, logging, and analysis tools available through MATLAB® and Simulink®, which go well beyond a logic analyzer offered by HDL simulators. One can simplify the analysis of large, complex models using model slicing [17] by focusing on interest areas based on functional dependencies determined via simulation or formal methods. This feature is also useful for interference analysis freedom in the system specification step, according to ISO 26262.

C. Continuous verification and validation

The main idea behind FPGA, SoC, and ASIC verification in MBD is first to verify model's behavior against its test requirements using static and dynamic methods. In the second stage, we automatically prove the behavioral equivalence between the model and the generated HDL code and its implementation on bits and cycle levels. The bulk of the verification effort is shifted towards the model level, where it is more directly linked to requirements, realizing the semiconductor industry's "shift-left" trend.

The first stage in this process deals with model-based verification and validation (model V&V), intending to demonstrate that the model used for production code generation behaves as specified by the requirements and that all requirements are implemented. Model V&V applies the same set of techniques on models that are mainstream for HDL code verification [5], namely formal methods and simulation-based techniques. Formal methods provided by Simulink® Design Verifier™ statically identify hidden design errors in models such as division by zero, overflows, and dead logic without using simulation. Formal property checking is also supported: one can describe safety requirements in a mathematical form, and the tool can prove or disprove compliance and generate counterexamples in case of the latter.

Simulink provides simulation-based verification through multi-domain, high-fidelity models of analog, mixed-signal, RF, and mechatronic components support. These models may be used to construct testbenches for algorithms designed for implementation in hardware and software. Model coverage is measured by several methods —execution, condition, decision, modified condition/decision (MCDC) — and can be highlighted in block diagrams and state machine models. This approach ensures test completeness and unintended functionality absence. An array of test tools is provided for creating test sequences and assessments for different use cases [18]. Constrained-random verification (CRV) can be applied to verify models before code verification [19]. Various static and dynamic assertion blocks also support assertion-based verification (AVB). Test harnesses can be automatically created to isolate components and apply a range of test scenarios. Test cases and their assessment must be linked with test requirements for measuring functional test coverage and essentially perform requirement-based testing. The V&V process can be centralized using the Test Manager [20], which can be coupled with continuous integration (CI) tools such as Jenkins [21]. While integrating components from different development branches, differences between related models can be displayed visually, and merging is possible. Testbenches and test harnesses are added successively for continuous verification and integration testing as designs progress through the development cycle. V&V tool usage can be qualified for ISO 26262 with a tool confidence level 2 (TCL2), and the IEC Certification Kit provides automated test suits for ease of tool qualification.

HDL code verification is the second stage. The goal is to demonstrate the equivalence of a Simulink specification model and the generated RTL through simulation result comparison. This step reuses the model testbenches automatically. It ultimately ensures that the code is acting according to the specification. This step reuses model testbenches in several automated actions. The first is HDL cosimulation with logic simulators such as Siemens® QuestaSim™ / ModelSim™, Cadence® Xcelium®, or Xilinx® Vivado® [13]. An

automatically generated cosimulation testbench applies input signals to the specification model and HDL code and compares their outputs. Second, VHDL, Verilog, and SystemVerilog DPI behavioral testbenches can be generated by comparing the equality and assessing the path/fail behavior with assertions. This method suits design teams running unit tests using logic simulators on server farms. Model-level assertions can also be converted into SystemVerilog assertions. Third, design teams can generate Universal Verification Methodology (UVM) environments or individual verification components from Simulink models [22]. FPGA-in-the-loop, a hardware-based method, may also verify the equivalence between the model and the FPGA implementation in an approach similar to HDL cosimulation. Code verification workflows can be added to the Test Manager for regression testing. Manually written HDL code can be integrated and verified with cosimulation and FPGA-in-the-loop or by leveraging the testbench export capabilities we just described.

Overall hardware integration testing is usually done using the hardware in the loop. By using Speedgoat™ target computers [23] — the same test harnesses and model tests created for simulation — can be reused to test the integrated system in real time. Plant models can be automatically deployed along with test cases for real-time simulation on the Speedgoat computer's CPU or optional FPGA boards. During the test execution, hardware board or ECU responses will be collected and sent back to the Test Manager for validation.

III. Conclusion

We presented the current trends in the automotive industry and the industry's challenges involving the design of FPGAs, programmable SoCs, and ASICs algorithm implementation. We noted the issues with conventional design flows and described how a model-based design approach could significantly improve product quality, shorten development time, reduce product costs, and enable more rapid responses in requirements or implementation target changes. The ISO 26262-qualified, integrated toolchain promotes multi-domain collaboration and ultimately helps streamline embedded hardware algorithm design and certification.

References

[1] O. Burkacky, J. Deichmann, J. P. Stein, " Outlook on the automotive software and electronics market through 2030," McKinsy&Company market analysis report

[2] Future Market Insights, "Automotive radar market outlook (2022-2032)," Future Market Insights market analysis report

[3] Future Market Insights, "Mobile LiDAR scanner market overview," Future Market Insights market analysis report

[4] D. Seidel "The growing use of FPGAs in motor control," in Electronic Products.

[5] H. Foster, "The 2022 Wilson Research Group Functional Verification Study", Siemens.com

[6] MathWorks, "Measuring the return on investment of Model-Based Design", White paper

[7] MathWorks, "System Composer," mathworks.com

[8] J. Krasner. "How product development organizations can Achieve long-term cost savings using Model-Based Systems Engineering (MBSE), " in Embedded Market Forecasters

[9] R. Aarenstrup, G. Tomar "Agile and Model-Based Design for Engineering Software Development," mathworks.com

[10] M. Adedjouma, N. Yakymets. "A Model-Based Safe-by-Design approach with IP reuse for automotive applications," ICSEA 2020 (2020): 122.

[11] MathWorks, "HDL Coder," mathworks.com

[12] MathWorks, "Fixed Point Designer" mathworks.com

[13] MathWorks, "HDL Cosimulation with MATLAB or Simulink" mathworks.com

[14] MathWorks, "IEC Certification Kit" mathworks.com

[15] MathWorks, "Hardware-Software Co-Design Overview" mathworks.com

[16] MathWorks, "SoC Blockset" mathworks.com

[17] MathWorks, "Simplify Model for Targeted Analysis of Complex Models Using Model Slicer Tool" mathworks.com

[18] MathWorks, "Assessments, Criteria, and Verification" mathworks.com

[19] MathWorks, "Add Random Constraints to Sequences in UVM Test Bench" mathworks.com

[20] MathWorks, " Test Manager" mathworks.com

[21] D. Boissy, P. Urban, K. Balasubramanian, P. R. Cumbreras, C. Branch, J. Pulipati "Continuous Integration for Verification of Simulink Models", mathworks.com

[22] MathWorks, " UVM Component Generation Overview" mathworks.com

[23] Speedgoat.com

[24] MathWorks, "Renesas Designs and Implements Image Processing IP Core for ASICs with Model-Based Design", mathworks.com

[25] A. Mauderer, J. H. Oetjens "System-level design of mixed-signal ASICs using Simulink: Efficient transitions to EDA environments" EETimes

[26] N. Kosugi, K. Hori, Y. Ishida, "Driving the Adoption of Model-Based Design for Communications System Development at Hitachi", mathworks.com