WHITE **PAPER**

cadence®

# Accelerate Algorithm to Silicon Development with Stratus HLS

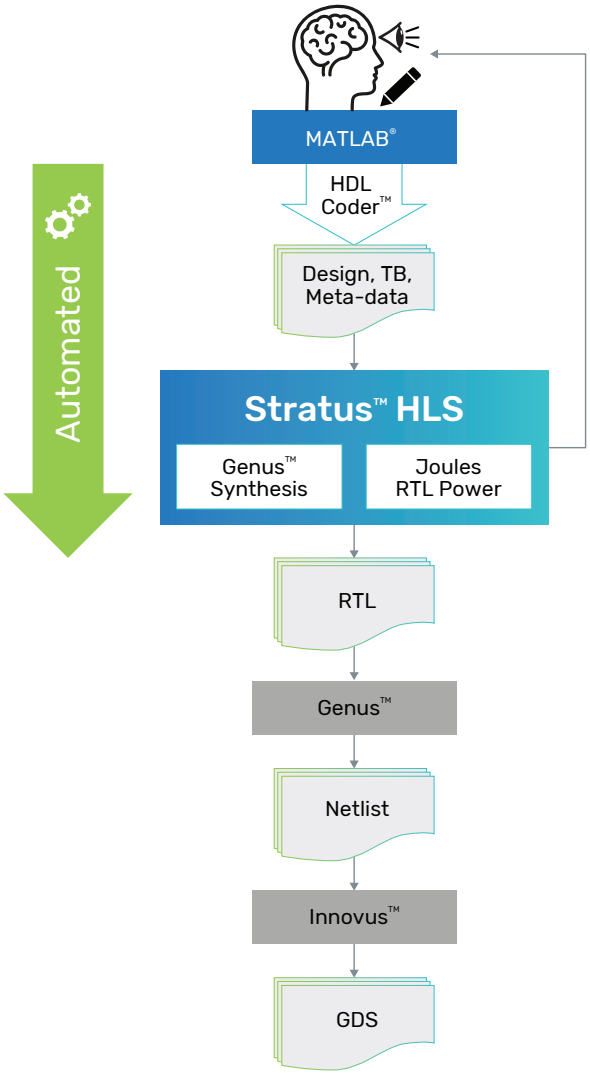
By Jeff Roane, Director of Product Management, and Vinod Kumar Khera, PhD

Growth in demand for artificial intelligence (AI) and digital signal processing (DSP) applications, coupled with advances in semiconductor process technology, drives increasingly denser SoCs. These complex SoCs further challenge the design team's ability to meet performance, power, and area (PPA) goals within tight time-to-market windows. We need automated and targeted solutions that efficiently handle algorithmic design content inherent in AI and DSP to keep up. Such solutions must also be well correlated with results obtained from physical design to minimize physical design iterations. Designers rely on several mature production-proven tools, from algorithm development to physical implementation. Any improvements to the design flow should maximally leverage existing tools where possible and deliver focused automation to address the challenges of iteration and poor correlation.

## Contents

## Algorithm Development

Algorithm development begins with mathematical models written in C++ or similar abstract programming languages. MathWorks MATLAB is the de facto industry-standard multi-paradigm programming language and numerically compute environment. In addition to algorithm performance tuning, the developers commonly make tradeoffs between numeric data types, algorithm fidelity, and PPA. For example, using single-precision floating-point data types may provide adequate algorithm fidelity and handle saturations and overflows, but representing data as a 32-bit IEEE 745 floating point is PPA expensive and may be overkill for the design requirements. Today, these tradeoffs are made without accurate PPA visibility because getting accurate PPA requires traversing the design implementation phase. Any improvement over the conventional flow should give algorithm developers early access to the implementation PPA without burdening the design implementation team. With such visibility, algorithm developers can tune their algorithms with the confidence that they have made data-driven PPA tradeoffs that correlate well with downstream implementation.




- Automated creation of synthesizable SystemC
- Gives architects accurate visibility into PPA envelope
- Automated import and Stratus project creation
- Architectural exploration and optimization
- Eliminates iteration
- Improves PPA
- Reduces schedule

*Figure 1: The integration allows algorithm developers to refine their algorithms with accurate data-driven PPA measurements that correlate well with downstream implementation*

## RTL Design Implementation

MATLAB is easy to use and contains comprehensive toolboxes for multiple application domains, so it is used widely for architectural exploration and algorithm development. However, we need register transfer language (RTL) for the hardware implementation of these algorithms in an IC or ASIC. So, it is required to convert the models implemented in high-level languages such as MATLAB, C/C++, and SystemC™ to RTL.

Moving from abstract MATLAB models to RTL descriptions has required manual conversion of MATLAB code into RTL. By definition, an RTL description expresses the cycle-accurate behavior. This step involves the design of a micro-architecture that accurately captures detailed cycle-by-cycle behavior and schedules operations among finite hardware resources to meet PPA goals in the implementation. To achieve optimal PPA, the micro-architectural solution space must be thoroughly explored—where the impact of alternate schedules, resource sharing, and data types can be measured. Unfortunately, the solution space is rarely explored because of the high engineering cost and time required to manually write RTL for candidate micro-architectures. An apparent deficiency of the conventional flow—where algorithm development is abstracted from implementation—is the absence of accurate PPA measurements to drive algorithm refinement, along with a lack of downstream visibility and increased risk of missing PPA goals.


*Figure 2: Stratus HLS takes an abstract design description in MATLAB, C, and C++ and optimizes the micro-architecture, yielding more optimal RTL that meets design constraints*

This is precisely the problem solved by high-level synthesis (HLS). Cadence® Stratus™ HLS is the market-leading HLS solution, as reported by Pedestal Research 2020 HLS market analysis. Stratus HLS takes an abstract design description in MATLAB, C, and C++ and optimizes the micro-architecture, yielding more optimal RTL that meets the design constraints. A key component of Stratus HLS is its programmable exploration environment, which enables automated exploration of candidate micro-architectures for selecting the most optimal micro-architecture. It is based on accurate measurements rather than inaccurate estimates.

Stratus HLS automates the design and verification flow from transaction-level modeling (TLM) to gates. The architecture exploration involves proposing changes to the design constraints, optimization, simulation, and results analysis to achieve the desired PPA. At a high level, it comprises the below exploration steps, yielding multiple architectures in an effort to find an implementation that best meets design requirements:

- Define and simulate the high-level description in C++ or SystemC
- Apply HLS and synthesis constraints for optimization and RTL generation
- Simulate to verify functionality and prepare for power analysis
- PPA analysis

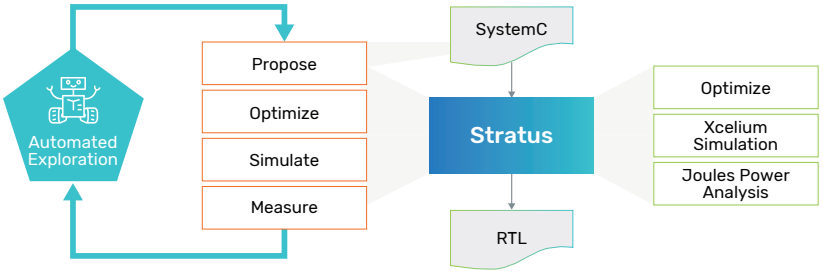



*Figure 3: The programmable exploration environment in Stratus HLS supports a rapid propose, optimize, simulate, and measure loop that automates micro-architecture exploration to achieve the desired PPA*

## Integrating MATLAB and Stratus HLS

Cadence and MathWorks developed the industry's first integration of a multi-paradigm numeric compute environment (MATLAB) and HLS (Stratus HLS) to address the challenges presented by the conventional flow where algorithm development and RTL implementation are disparate activities.

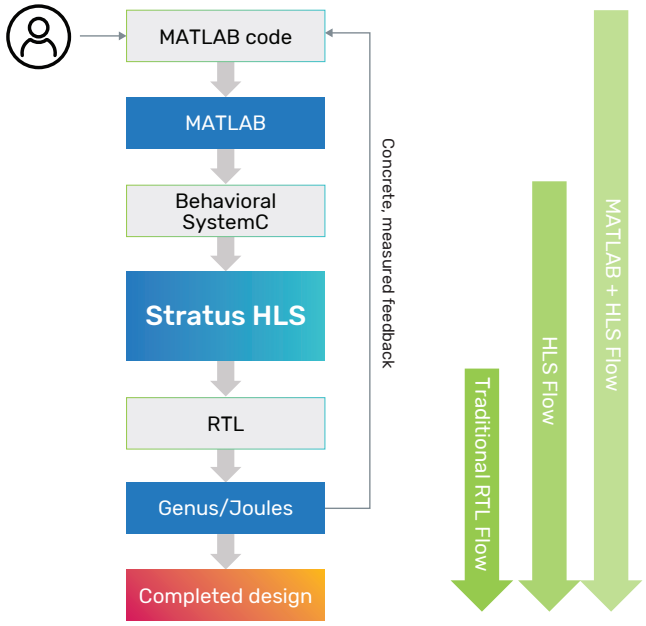



*Figure 4: MATLAB and Stratus HLS address challenges presented by the conventional flow where algorithm development and RTL implementation are disparate activities*

The integration first automates the handoff of the algorithm described in MATLAB to Stratus HLS via MATLAB's HDL Coder option. HDL coder translates the MATLAB code description into synthesizable C++ that Stratus HLS can read. This handoff includes the design, testbench (input data and expected response), and meta-data, allowing automated project creation by Stratus HLS. By having the design and testbench handed off together, design verification and activity-driven power analysis can be performed on the resultant RTL produced by Stratus HLS.

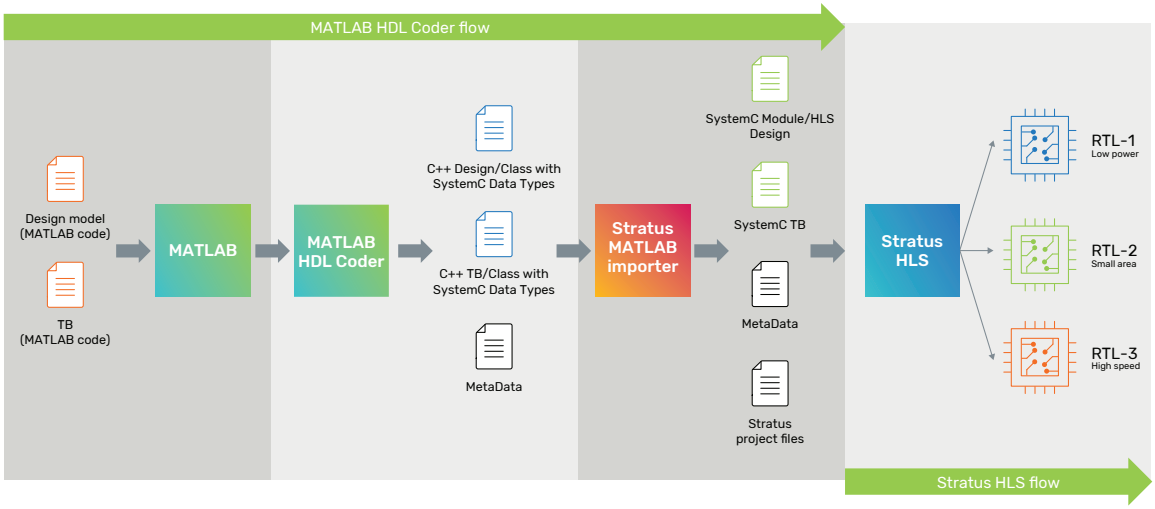
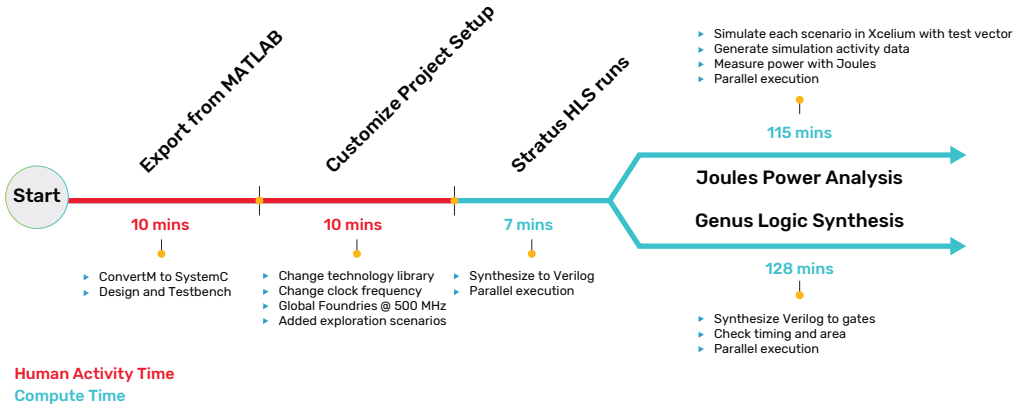



*Figure 5: The automated handoff of an algorithm described in MATLAB to Stratus HLS via MATLAB's HDL Coder option*

The integration squarely addresses the challenges of the conventional flow by providing early and accurate PPA measurements for the algorithm development team and the automated production of optimal RTL for the design implementation team. Stratus HLS is designed to automate and accelerate design by moving manual work to a higher level of abstraction. Together algorithms can be tuned to yield more optimal PPA, and the solution space can be explored and optimized, ensuring that the design implementation team achieves the same PPA viewed by the algorithm developers.

## AES Encryption Example

To better understand the productivity gain of this integration, consider an example where an advanced encryption standard (AES) encryption algorithm is initially developed in MATLAB, then exported to Stratus HLS. Wall clock times are measured for each step. Figure 6 shows the time consumed by each step. It shows the process, with steps requiring human activity in red and fully automated compute-time steps in blue.




| Pipeline Throughput | MATLAB SystemC | |
|---|---|---|
| | Area (µm²) | Power (mW) |
| 1 | 19.475 | 10.7 |
| 2 | 13.194 | 10.1 |
| 4 | 9.877 | 7.5 |
| 6 | 8.139 | 5.5 |
| 11 | 6.339 | 4.5 |

Total Elapsed Time = 153 mins

**For this example, scenarios, 20 mins human-time, 2 hours compute time!**

*Figure 6: Testcase showing the productivity gain delivered by the integration—five micro-architectures explored with only 20-minutes of human activity time and 2 hours of compute time*

For this example, the user could explore five micro-architectures with only 20 minutes of human activity time and two hours of compute time. Significantly less than what would be required in a conventional manual flow—such a flow would take weeks and a small team of engineers. These steps are detailed below:

1. **Export the MATLAB:** After developing and simulating the algorithm in MATLAB, the user exports the design using MATLAB's HDL Coder. HDL Coder translates the MATLAB code into synthesizable C++. HDL Coder also exports the testbench and metadata. The testbench is composed of stimulus and expected results.
2. **Customize the Setup:** Stratus HLS' Behavioral Design Workbench (BDW) import function reads the design, testbench, and metadata and creates a Stratus HLS project. The user configures the Stratus HLS project by specifying the technology library, design constraints, and exploration settings.
3. **Exploration:** Once configured, Stratus HLS explores the solution space. Exploration is governed by a Stratus HLS Tcl control file that defines how constraints are changed and imposed on the candidate micro-architecture designs.
4. **Analysis:** In addition to HLS optimization, Stratus HLS launches Cadence's Genus™ Synthesis Solution, Xcelium™ Logic Simulator, and Joules™ RTL Power Solution to verify functionality and measure power consumption.

## Conclusion

In a conventional design flow, getting accurate implementation PPA details during the algorithm development phase can be very challenging, if not impossible, as it requires burdening the implementation team with executing trial implementations. Due to this steep challenge, algorithm developers use imprecise estimates for PPA prediction. The imprecise estimates used by algorithm developers often result in a failure to meet the PPA goals. The MATLAB integration with Stratus HLS gives algorithm developers rapid access to accurate PPA details, allowing measurement-driven algorithm refinement. The solution minimally disturbs the existing design flow and automates much of the manual process from MATLAB to implementation details. This integration also automates micro-architectural exploration and the production of more optimal RTL, which yields improved PPA and shorter design schedules. Early results suggest that this flow will become the de facto standard method of developing and implementing algorithms targeted for silicon.

cadence®

Cadence is a pivotal leader in electronic systems design and computational expertise, using its Intelligent System Design strategy to turn design concepts into reality. Cadence customers are the world's most creative and innovative companies, delivering extraordinary electronic products from chips to boards to complete systems for the most dynamic applications. **www.cadence.com**