**Product how-to: Achieving STARC and DO-254 compliance using HDL Coder-generated code**

*kiran kintali* - December 09, 2014

**HDL Coder** generates target independent, synthesizable Verilog and VHDL code from **MATLAB** functions, **Simulink** models, and **Stateflow** charts. The generated HDL code is bit-true and cycle-accurate to source designs and can be targeted to FPGAs and ASICs for both prototyping and production. This article is designed to provide a basic understanding of the main concepts of achieving compliance with Semiconductor Technology Academic Research Center (STARC) design style guides and the DO-254 standard using HDL Coder. We outline major steps involved in generating standards-compliant ASIC/FPGA design using HDL Coder. We also explain how HDL Coder features can be used to enhance and facilitate critical stages of compliance testing in your hardware design workflow.

DO-254 and STARC (see sidebar) are popular standards in North America and Japan. Compliance with these standards involves processes more rigorous than standard ASIC/FPGA design and verification flows. These standards and the associated processes ensure design quality.

Development processes used for industrial Field Programmable Gate Array (FPGA) and Application Specific Integrated Circuit (ASIC) applications, such as DO-254 in the aerospace industry, may recommend the use of certain RTL coding guidelines.

This article focuses on the following three key aspects of HDL code generation using HDL Coder that attempt to simplify the workflow aspects of DO-254 and STARC:

- Design rules
- Coding customizations
- Traceability

RTL checking is an integral part of ASIC/FPGA design team workflows. Design teams define a set of rules for writing RTL that ensures code quality. For example, when manually inspecting their code, design teams check for common coding errors, such as incomplete FSM, latches, and incomplete sensitivity lists. Design teams use industry-standard lint tools and other design rule checkers to catch these errors automatically.

HDL Coder and **Model-Based Design** simplify achieving compliance by generating RTL that is correct by construction. RTL rules from many standards are usually specified at the HDL level, but HDL Coder provides a model-level granularity of the RTL rules as it applies to MATLAB and Simulink designs. It also automates the process of checking the generated RTL using HDL lint or compliance validation tools with turnkey script generation.
A typical SoC design usually contains many IP cores developed by multiple teams. Establishing unified design rules for coding style is key for accurate simulation and synthesis, and it improves the readability and maintainability of RTL. The quality of the IP is critical for such a complex system during integration. RTL code generated using Model-Based Design and HDL Coder ensures that these guiding principles are met.

**HDL Coding Standards**

HDL Coder seeks to generate VHDL and Verilog code that meets common industry coding guidelines such as DO-254, STARC, and RMM. HDL Coder generates reports that help engineers identify unsuitable constructs in Simulink models and MATLAB code so engineers can adapt their models to produce generated RTL that follows these coding guidelines.

Code generated using HDL Coder follows RTL coding principles by:

- Avoiding FSM state reachability and coding issues
- Avoiding differences between simulation and synthesis semantics
- Avoiding operations with expensive implementation costs
- Avoiding downstream tool flow issues
- Following naming and RTL coding conventions in the generated code
- Enforcing RTL modeling clarity and reducing complexity
- Enforcing checks for clock bundles (clocks, enables, resets) and control signals
- Enabling testability and traceability of the code

Here is a classification of coding rules supported by HDL Coder in the generated RTL.

![Figure 1. Coding standard rule categories in HDL Coder](image)

**Generate an HDL Coding Standard Report**

To generate an HDL coding standard report using the command line interface, set the HDLCodingStandard property to Industry when using the makehdl command to generate code from a model.

**HDL Coding Standard Report**

The HDL coding standard report shows that a generated HDL code conforms to an industry coding standard selected when generating code.
The report points to problematic modeling issues in MATLAB code and Simulink models that can lead to coding standard compliance errors, warnings, and messages in the generated RTL. Errors and warnings in the report link to elements in the original design so that engineers can fix problems and then regenerate code. Messages show where the coder automatically corrected the code to conform to the coding standard.

The report also lists the rules in the coding standard with which the generated code complies. Engineers can inspect the report to see which coding standard rules the coder checks.

![Figure 2. Coding Standard report for Simulink model 'demo_double/Subsystem'](image)

**Rule Summary**

The rule summary section shows the total numbers of errors, warnings, and messages, and lists the corresponding rules. Each rule shown in the summary links to the rule in the detailed rule hierarchy section.

**Rule Hierarchy**

The rule hierarchy section lists every rule that the coder checks within these three categories:

- Basic coding practices, including rules for names, clocks, and reset.
- RTL description techniques, including rules for combinatorial and synchronous logic, operators, and finite state machines.
- RTL design methodology guidelines, including rules for ports, function libraries, files, and comments.

**HDL Coding Standard Rules**

When engineers generate a HDL coding standard report, the following table shows examples of industry standard rules that may appear due to problematic constructs in their model. The errors
can be fixed or warnings related to these rules by updating the Simulink model or code.

<table>
<thead>
<tr>
<th>Rule/Severity</th>
<th>Message</th>
<th>How to fix</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.A.3.5 Error</td>
<td>Do not use case variants of names in the same scope. (Verilog) Do not use names that differ in case only, within the same scope. (VHDL) Two or more names in your design, within the same scope, are identical or scope free. For example, the names <code>Ground</code> and <code>Power</code> cannot be in the same scope.</td>
<td>Update the names in your design so that no two names within the same scope differ only in case.</td>
</tr>
<tr>
<td>1.A.3.6 Warning</td>
<td>Top-level module/entity and port names should be less than or equal to 16 characters in length and not be mixed-case. A top-level module, entity, or port name in the generated code is longer than 16 characters, or uses letters with mixed case.</td>
<td>Update the indicated name in your design so that is less than or equal to 16 characters long, and that a letter is lowercase or all letters are uppercase.</td>
</tr>
<tr>
<td>1.A.3.7 Error</td>
<td>Module and instance names should be between 2 and 32 characters in length. (Verilog) Entity names and instance names should be between 2 and 32 characters in length. (VHDL) A module, instance, or entity name in the generated code is less than 2 characters long, or more than 32 characters long.</td>
<td>Update function names or subsystem names in your design to be between 2 and 32 characters long.</td>
</tr>
<tr>
<td>1.A.3.8 Error</td>
<td>Signal names, port names, parameter names, define names and function names should be between 2 and 40 characters in length. (Verilog) Signal names, variable names, type names, label names and function names should be between 2 and 40 characters in length. (VHDL) A signal, port, parameter, define, or function name in the generated code is less than 2 characters long, or more than 40 characters long.</td>
<td>Update the indicated name in your design so that is between 2 and 40 characters long.</td>
</tr>
<tr>
<td>1.A.3.9 Error</td>
<td>Include files must have extensions that match &quot;.hd&quot;, &quot;.vh&quot;, &quot;.inc&quot;, and &quot;.vh&quot;, &quot;.inc&quot;, &quot;.tck&quot; for testbench. (Verilog) Package file name should be followed by &quot;.vhdl&quot;. (VHDL) The filename extension of an include file is not one of the standard extensions.</td>
<td>Set the Verilog file extension or VHDL file extension to one of the standard extensions. Use the <code>VerilogFileExtension</code> and <code>VHDLFileExtension</code> properties from the command line.</td>
</tr>
<tr>
<td>2.C.1 Error</td>
<td>Do not specify flip-flop (or RAM) initial value using initial construct. The generated HDL code for your design contains an unsynthesizable initial statement.</td>
<td>Disable the Initial block RAM or Initializer RAM blocks option in the EDA Workflow Advisor.</td>
</tr>
<tr>
<td>2.E.15 Warning</td>
<td>Large multipliers must not be described using the multiplication operator with RTL. The generated HDL code contains a multiplication operator (*) where the output of the multiplication has a bitwidth of 16 or greater.</td>
<td>In your design, implement multiplications by use of a shift-and-add algorithm, or ensure that the data size of the output of a multiplication does not require a bitwidth of 16 or greater.</td>
</tr>
<tr>
<td>3.B.8.1 Error</td>
<td>Non-integer type used in the declaration of a generic may be unsynthesizable.</td>
<td>Modify your design to use fixed-point data type.</td>
</tr>
</tbody>
</table>

Customizations to Coding Standard Report and Conformance Checks
HDL Coder allows customization of several coding standard rules using the 'HDLCodingStandardCustomizations' property-value settings or the GUI pane in the MATLAB or Simulink workflow advisors.

**Generate an HDL Lint Tool Script**
HDL Coder also generates third-party lint tool scripts that can be used to check the generated HDL code in with compliance validation tools. Engineers can also configure script generation to generate a lint script for a custom tool or standard. The software writes the initialization, command, and termination strings to a script that can be used to run the third-party tool.
Using HDL Coder is beneficial in workflows for DO-254 and for following STARC design style where
it is important to generate code that conforms to industry coding standards. Model-Based Design allows you to model your algorithms in MATLAB and Simulink and generate RTL code using HDL Coder. HDL Coder features can be used to enhance and facilitate critical stages of compliance testing in the hardware design workflow. Engineers can avoid common coding errors, customize generated RTL code using HDL Code generation features, detect and avoid modeling issues that can lead to design rule violations, and integrate with design rule checkers to check the validity of the generated RTL code.

Sidebar: Coding standards

Some of the RTL coding standards that HDL Coder generated code is built on:

- **DO-254:** The DO-254 standard was formally recognized by the FAA in 2005 via AC 20-152 as a means of compliance for the design of complex electronic hardware in airborne systems. Complex electronic hardware includes devices like Field Programmable Gate Arrays (FPGA), Programmable Logic Devices (PLD), and Application Specific Integrated Circuits (ASIC).

- **STARC:** Semiconductor Technology Academic Research Center (STARC) policy guidelines are an extensive set of rules that ASIC and system on chip (SoC) designers use to perform in-depth structural analysis on Verilog and VHDL Register Transfer Level (RTL) descriptions. STARC guidelines are compiled by consortium of 11 major Japanese semiconductor companies that promote a design standard for IP trade and reuse.


Also see:

- [Model Based Design for FPGA Development](#)
- [Early verification and validation using model-based design](#)
- [Model-based design test tools hasten development](#)