

Automating Code Reviews with Simulink Code Inspector

Mirko Conrad, Matt Englehart, Tom Erkkinen, Xiaocang Lin, Appa Rao Nirakh,
Bill Potter, Jaya Shankar, Pete Szpak, Jun Yan, Jay Clark

The MathWorks, Inc., Natick, MA, USA

Abstract: Safety standards such as DO-178B require source code reviews. Given the maturity of today's code generators, the effectiveness of manual reviews of automatically generated code is rather limited. This results in a strong desire to automate reviews of automatically generated code. This paper introduces Simulink Code InspectorTM, a novel tool to automate manual code reviews of C source code generated from Simulink® models.

1 Code Reviews according to DO-178B

DO-178B [DO-178B] is a certification standard for civil aviation published in 1992. It provides “guidelines for the production of software for airborne systems and equipment that performs its intended function with a level of confidence in safety that complies with airworthiness requirements”. To detect and report errors that might have been introduced during the software coding process, DO-178B requires reviews and analyses to “confirm that the outputs of the software coding process are accurate, complete and can be verified” (cf. DO-178B, section 6.3.4). The objectives of software code reviews are detailed in DO-178B, table A-5. A common approach to satisfy these objectives is to carry out manual reviews of the source code. Manual code reviews are labor intensive and in case of automatically generated code typically not very effective. Practitioners estimate that about 50 lines of source code (LoC) can be reviewed in a one hour period, resulting in engineering costs of \$150. Projected onto a 100.000 LoC project, the source code review would last 2000 hours and cost \$ 0.3 million. Given the maturity of today's code generators, the efficacy of manual review of automatically generated code is quite limited. [Pot04] e.g. reports about the certification of DO-178B flight code at Honeywell. The amount of code certified exceeded 1.000.000 LoC per year. The study reports that only one code generation error was found in a one year time period. Using the above cost estimates, this would translate to a cost of \$ 3million to find one bug. These numbers speak for themselves. The low efficacy of manual reviews of generated code creates a high demand for automating the code review process.

2 Simulink Code Inspector Overview

To automate reviews for source code developed from Simulink models, the authors developed Simulink Code Inspector [SLCI] a novel tool to automate code reviews required by DO-178B and other functional safety standards. Simulink Code Inspector carries out a translation validation of C code generated from a Simulink [Simulink] model using the Embedded CoderTM [ECoder] code generator. In particular, SLCI systematically examines blocks, parameters, and settings in a Simulink model to determine whether they are structurally equivalent to operations, operators, and data in the C source code generated from the model (Fig. 1). Inputs to Simulink Code Inspector are a Simulink model and the C source code generated by the Embedded Coder code generator for this model. SLCI processes these two inputs into internal representations (IRs), called model IR and code IR. These IRs are transformed into normalized representations to facilitate further analysis. In this process, the model IR represents the expected pattern, whereas the code IR constitutes the actual pattern to be verified. To verify the generated code, SLCI attempts to match the normalized model IR with the normalized code IR. In this process SLCI examines aspects such as model interface, block behavior, block connectivity, execution order, data / file packaging, and usage of local variables.

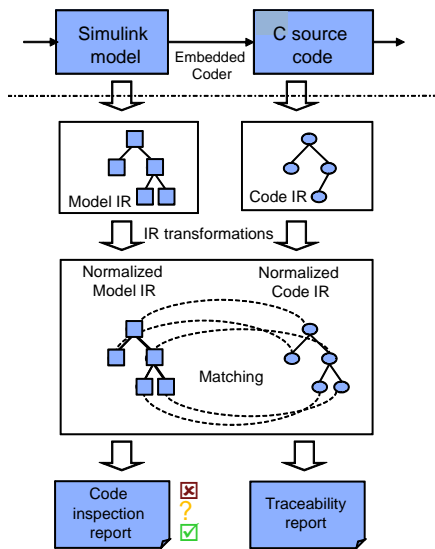


Figure 1. Code Inspection Approach

The utilization of normalization techniques allows to inspect code generated by a highly-optimizing code generator. The results of this matching process are reported to the user by means of a verification report and a traceability report. In case of a completed code inspection, the verification report documents the translation validation process whereas the traceability report maps the model elements onto their counterparts in the generated code and vice versa. The reports generated by Simulink Code Inspector are used as evidence to document the automated code review process. Upon DO-178B verification tool qualification, SLCI significantly reduces time and cost associated with verifying code against requirements. Instead of completing manual line-by-line code reviews with a project checklist, which is time intensive and error prone, users can run the Code Inspector and review a detailed inspection report.

3 Example: Roll Axis Autopilot Model

In this section we illustrate the usage of Code Inspector by using the example of a Roll Axis Autopilot model and the C code generated from this model.

Simulink Code Inspector can be invoked from a graphical user interface or via a MATLAB command line API. To kick-off the code inspection, the user can launch a compatibility checker that helps to identify modeling constructs and tool settings that are not supported by the Code Inspector. The compatibility checker leverages the Model Advisor infrastructure [MdlAdv] to statically analyze the model. If applicable, the compatibility checker also provides suggestions on how to replace incompatible modeling constructs with ones supported by the tool.

After passing the compatibility check, the user can initiate the actual code inspection process. The tool can be configured to check an individual model or an entire hierarchy of referenced models. Fig. 2 (left) shows the top level of the Simulink model of a roll axis auto pilot representing the low-level requirements for the auto pilot functionality. In this example, the auto pilot functionality is represented by three different models, the top level model and separate referenced models for the HeadingMode and BasicRollMode calculations.

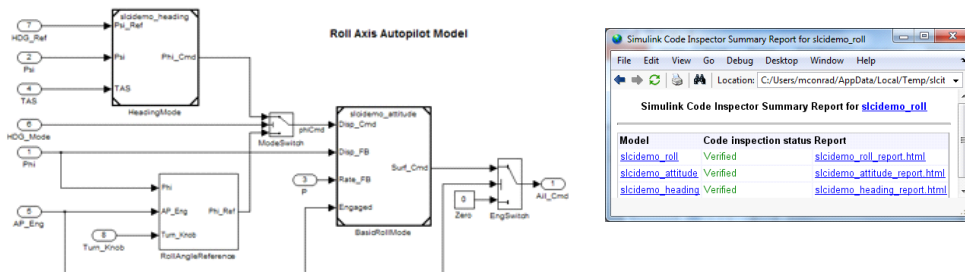


Figure 2. Roll Axis Autopilot Model – Top Level (Left) and Code Inspection Overview (Right)

If the code has been generated previously, it can be retrieved from the configuration management system. Alternatively it can be generated on-the-fly prior to the actual code inspection process. The code inspection results are presented in a hierarchical manner. An overview report provides aggregated status information and links to the detailed results for each model in the hierarchy (Fig. 2, right). The detailed results are divided into two parts, verification report (Fig. 3) and traceability report (Fig. 4).

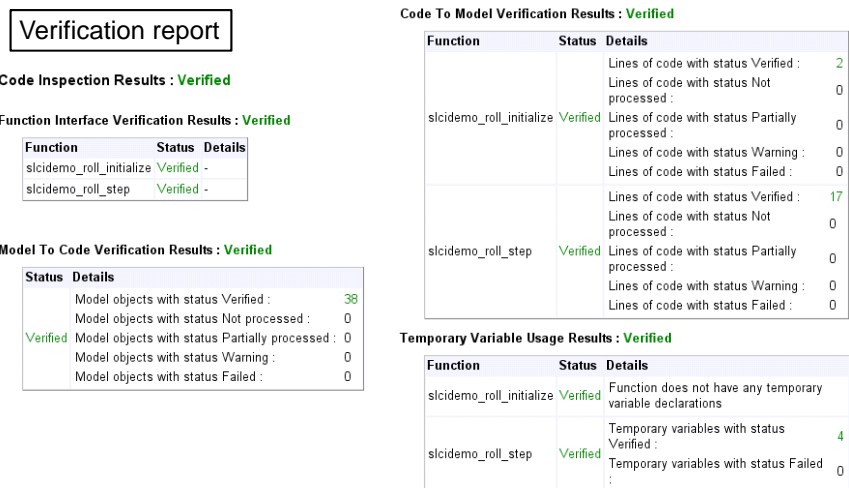


Figure 3. Code Inspection Report – Verification Report for the Top-level System

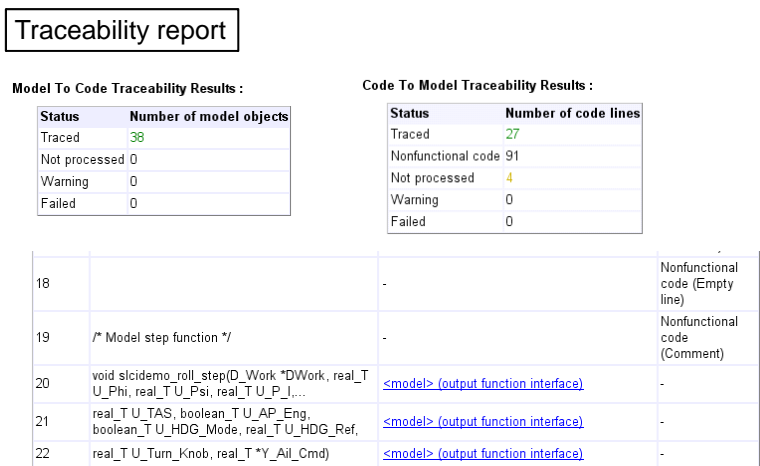


Figure 4. Code Inspection Report – Traceability Report for the Top-level System

The verification report features information about verification of the interfaces of generated code functions, verification of structural equivalence between model and code, and usage of temporary variables. The example in Fig. 3 shows a result for generated code that is structurally equivalent to its corresponding model. Model elements that are outside of the supported language subset and corresponding code fragments would be indicated as ‘not processed’ in the verification report. The traceability report documents the code lines that implement a particular model element and the model elements that contributed to the generation of a line of code. The lower part of Fig. 4 shows parts of code-to-model traceability information provided by SLCI.

To illustrate the error detection capabilities of Simulink Code Inspector, we'll intentionally modify the generated code (i.e. carry out a code mutation) to simulate a bug in the generated code. Then we'll carry out a code inspection of the original model against the modified code. This process is illustrated in Fig. 5. The upper left part of the figure shows a lower level of the RollAngleReference subsystem model. The algorithm contains an OR block indicated using red color. The upper right part of Fig. 5 shows the line of code implementing the OR block. To simulate a code generation bug, we create a code mutation that replaces the '||' operator with the '&&' operator. The lower parts of the figure show a section of the model-to-code verification results for the corresponding model. The report indicates that the matching process was not successful, i.e. Simulink Code Inspector did not find an expected code pattern for the OR block.

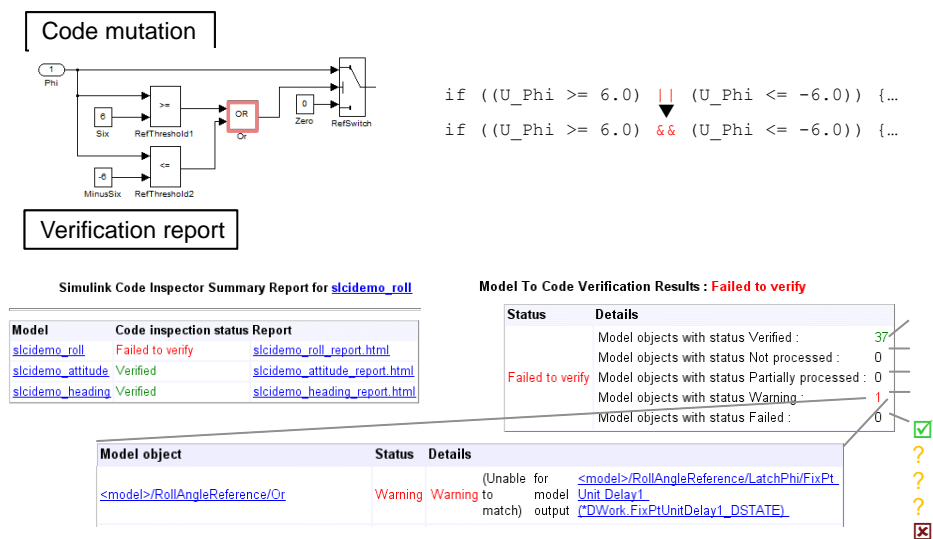


Figure 5. Code Mutation and Resulting Code Inspection Report

The auto pilot model used in this example is relatively small; the 3 model hierarchy consists of about 70 blocks and generates about 70 lines of C source code (without comments). The actual code inspection takes less than 30 seconds on a Windows 7 computer with an 1.73GHz Intel® Core™ i7 CPU and 8GB RAM. The approach used by Code Inspector is scalable and has been applied to verify complex, real world aerospace models that are 10x larger than the example model presented here.

4 DO-178B Tool Qualification and Certification Credits

The DO-178B tool qualification process is described in section 12.2 "Tool Qualification" of the standard [DO-178B] and has been further clarified by FAA order 8110.49 "Guidelines for the Qualification of Software Tools Using RTCA/DO-178B" [FAA 8110.49]. Tool qualification is needed when processes of DO-178B are eliminated, reduced or automated by the use of a software tool without its output being verified as specified in section 6 of DO-178B. The objective of the tool qualification process is to ensure that the tool provides confidence at least equivalent to that of the process(es) eliminated, reduced or automated [DO-178B]. DO-178B provides different tool qualification approaches for development tools (e.g. code generator, compiler) and verification tools (e.g. static analyzer, test tool).

Since the Code Inspector output is not part of the airborne software (i.e. SLCI cannot introduce an error into the software), the verification tool qualification process applies. To qualify a verification tool such as Simulink Code Inspector, it needs to be demonstrated that the tool complies with its Tool Operational Requirements under normal operational

conditions. This can be achieved by documenting the different verification capabilities (i.e. operational requirements) of SLCI and providing a test suite to demonstrate that the tool behaves as specified for each of these verification capabilities. The test suite would comprise of positive test cases (i.e. matching model-code pairs) and negative test cases (i.e. model-code pairs where model or generated code were subjected to a mutation).

Upon qualification of Simulink Code Inspector an applicant could claim certification credits for the applicable objectives for DO-178B, Table A-5. Fig. 6 lists the anticipated certification credits for Simulink Code Inspector in combination with the Polyspace® code verifiers [Polyspace] which provide complementary source code verification capabilities [SLCI]. As an example, SLCI verifies the code execution order against Simulink block execution order and the initialization of variables to show objective (2), i.e. that the source code complies with the software architecture. Details about the assumed software development and verification workflow as well as further information on the certification credits can be found in [DO Workflow].

Annex A Table	Objective	DO-178B Ref.	Software Levels	Anticipated Certification Credit [Tool(s)]
A-5	(1) Source Code complies with low-level requirements	Section 6.3.4a	A, B, C	Full [Simulink Code Inspector]
A-5	(2) Source Code complies with software architecture	Section 6.3.4b	A, B, C	Full [Simulink Code Inspector]
A-5	(3) Source Code is verifiable	Section 6.3.4c	A, B	Full [Simulink Code Inspector]
A-5	(4) Source Code conforms to standards	Section 6.3.4d	A, B, C	Full [Polyspace MISRA-AC ACG rules checker]
A-5	(5) Source Code is traceable to low-level requirements	Section 6.3.4e	A, B, C	Full [Simulink Code Inspector]
A-5	(6) Source Code is accurate and consistent	Section 6.3.4f	A, B, C	Full (for source code based criteria) [Simulink Code Inspector, Polyspace verifier]

Figure 6. Anticipated certification credits acc. to DO-178B, Table A-5

Code Inspector has been implemented independent of the Embedded Coder code. In particular, the IRs used by Code Inspector are different from the IRs used by the code generator. Further details on independence between Code Inspector and other Model-Based-Design tools are discussed in [DO Workflow].

5 Related Work and Discussion

Other translation validation tools for code generated from Simulink models include TVS [RS09, Rya09] and ClawZ [ACO+00]. The TVS translation validation tool for the Real-Time Workshop code generator utilizes an induction-based proof strategy to verify C code generated from Simulink models using the Real-Time Workshop Code generator. According to [RS09] the largest model automatically verified using TVS comprised about 100 blocks and resulted into ~250 lines of C source code. ClawZ is a prototype tool to translate a Simulink model into a Z specification. It can be used in combination with the Compliance Tool component of ProofPower to formally prove the compliance between the Z specification and an implementation in ADA [ACO+00]. It had been used to verify a flight control system comprising of 37 pages of diagrams against 1200 lines of ADA code.

The static translation validation approach implemented by Simulink Code Inspector that establishes structural equivalence between models and generated code can be complemented by dynamic, testing-based translation validation techniques to assess the numeric equivalence between models and generated code (back-to-back testing) [Con11]. A verification and validation workflow that combines automated code reviews with other verification and validation activities can be found in [DO Workflow].

Simulink Code Inspector supports a subset of the Simulink language. If the inspected model contains modeling constructs outside of the supported subset, SLCI won't verify the code for these model elements. To partially support models containing unsupported modeling features, such features can be wrapped with model blocks (model referencing). In such a case SLCI will be able to verify all model blocks not containing unsupported model elements. The remaining model blocks are subject to manual code reviews. Simulink Code Inspector will also verify the data and control flow between the model blocks.

6 Summary and Conclusion

Simulink Code Inspector is a novel translation validation tool for the Embedded Coder code generator that translates Simulink models into optimized C code. A first version was made available in 2011 as part of the R2011b release of the Simulink product family. By assessing the structural equivalence between a Simulink model and its corresponding C source code and providing bi-directional traceability information between the model and the generated code, Simulink Code Inspector automates code reviews in a Model-Based Design process. Code Inspector has been implemented independent of the Embedded Coder code generator and can be qualified as a verification tool according to DO-178B.

The tool automates code review activities required by DO-178B to demonstrate that the generated source code traces to and complies with the model. It provides benefits similar to a qualified code generator while allowing users to take advantage of state-of-the-art code generation with the Embedded Coder. The automation of labor-intensive code review processes facilitates significant cost savings. Simulink Code Inspector was developed in close cooperation with leading aerospace companies and verifies complex, real world aerospace models.

References

- [ACO+00] R. Arthan, P. Casely, C. O'Halloran, A. Smith: ClawZ – Control laws in Z. 2000
- [Con11] M. Conrad: Testing-Based Translation Validation of Generated Code. In: J. Zander, I. Schieferdecker, P. J. Mosterman (Eds.): Model-Based Testing for Embedded Systems. CRC Press, 2011.
- [DO-178B] RTCA/DO-178B "Software Considerations in Airborne Systems and Equipment Certification", RTCA SC-167 / EUROCAE WG-12, 1992
- [DO Workflow] DO Qualification Kit: Model-Based Design Workflow for DO-178B. The MathWorks, Inc. 2012.
- [ECoder] Embedded Coder™ product page. The MathWorks Inc., www.mathworks.com/products/embedded-coder
- [FAA 8110.49] FAA Order 8110.49 "Guidelines for the Qualification of Software Tools Using RTCA/DO-178B", FAA 2003
- [MdlAdv] Model Advisor. blogs.mathworks.com/seth/2008/11/04/introduction-to-model-advisor/
- [Polyspace] Polyspace® for C/C++ product page. The MathWorks Inc., www.mathworks.com/products/polyspace
- [Pot04] B. Potter: Use of The MathWorks Tool Suite to Develop DO-178B Certified Code. Embry Riddle Aeronautical University/ FAA Software Tools Forum, Daytona Beach, FL, USA, 2004
- [RS09] M. Ryabtsev, O. Strichman: Translation Validation: From Simulink to C. LNCS 5643/2009
- [Rya09] Ryabtsev, M.: Translation validation: From Simulink to C. Technical Report IE/IS-2009-01, Industrial Engineering, Technion, Haifa, Israel, 2009
- [Simulink] Simulink® product page. The MathWorks Inc., www.mathworks.com/products/simulink
- [SLCI] Simulink Code Inspector product page. The MathWorks Inc., www.mathworks.com/products/simulink-code-inspector