Production Code Generation using Model Based Design

Manohar Reddy
Application Engineering
MathWorks India Pvt Ltd
Agenda

- **Design for Embedded Implementation**
  - Code Generation Settings
  - Embedded Coder® Quick Start
  - Code Generation Advisor
  - Algorithmic Code and Full Executable Generation
  - Traceability (Model $\leftrightarrow$ Code)
  - MISRA C: 2012 Support

- **Automating Code Reviews for High Integrity Standards**

- **Generating AUTOSAR-compliant code from Simulink® models**
Code Generation Settings

- Simulation time:
  - Start time: 0.0
  - Stop time: 10.0

- Solver options:
  - Type: Fixed-step
  - Solver: discrete (no continuous states)
  - Fixed-step size (fundamental sample time): 0.01

- Tasking and sample time options:
  - Periodic sample time constraint: Unconstrained

- System target file: ert.tlc
  - Language: C
  - Description: Real-Time Workshop Embedded C
  - Target hardware: None

- Build process:
  - Toolchain settings:
    - Toolchain: Automatically locate an installed toolchain
Embedded Coder Quick Start

Select your most important code generation objective.

- Execution efficiency
- RAM efficiency

What to consider
Based on your selection, the Quick Start tool configures your model with the best optimizations for your specified code generation objective.

After Quick Start code generation is complete, you can fine-tune your optimization settings using the Code Generation Advisor.

About the selected option
If faster execution is your primary objective, select this option. The Quick Start tool configures the model to optimize code for execution speed and RAM usage. The priority...
Algorithmic Code Generation

Controller Model

Generated Algorithm Code

Input Drivers

Comm Drivers

Output Drivers

Special Device Drivers

Included Legacy Code

Scheduler/Operating System and Support Utilities

Communication Interfaces

Drivers

Special Interfaces

Tuning

Actuators

Communication Interfaces

Sensors

Input Drivers

Special Device Drivers

Scheduler/Operating System and Support Utilities
Code Generation Advisor
/** Model step function */
void Controller_step(void)
{
    int16_T rtb_Sum1;
    int16_T rtb_DataTypeConversion1;
    int32_T tmp;

    /* ModelReference: '<S3>/relative_threshold' */
    RelativeThreshold(&in_rotation, sController_DWork.threshold);

    /* Product: '<S3>/limit_ratio' */
    Controller_DWork.limit_ratio = (int16_T)div_nzp_s32_floor
            (Controller_DWork.threshold, Controller_DWork.threshold);

    /* sum: '<S3>/Merge' incorporates:
    * Inport: '<Root>/Reg Pressure'
    */
    rtb_Sum1 = (int16_T)(Controller_DWork.limit_ratio + in_pressure);

    /* Outport: '<Root>/pos_cmd' */
    Controller_Y.poscmd = rtb_Sum1;

    /* Outport: '<Root>/Fault1' incorporates:
    * Inport: '<Root>/Reg Pressure'
Code Metrics

Auto generated code is smaller than hand code

Table 2 shows ROM and RAM comparisons between hand code and auto code for a floating-point component in some typical powertrain software.

<table>
<thead>
<tr>
<th></th>
<th>Hand Code</th>
<th>Auto Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>ROM</td>
<td>6408</td>
<td>6192</td>
</tr>
<tr>
<td>RAM</td>
<td>132</td>
<td>112</td>
</tr>
</tbody>
</table>

The auto code has less size of ROM and RAM compared to that of hand code. The auto code is readable and peer reviewed, and checked with the QAC static analysis tool. Most importantly, the auto code is implemented in a real-world powertrain application.

CONCLUSION

A custom data class allowing data type and data scaling information to be incorporated into the model is
Full Executable Generation

Controller Model w/Driver Blocks

Communication Interfaces

Generated Algorithm Code

Input Drivers

Optional Target Optimized Code

Output Drivers

Special Device Drivers

Scheduler/Operating System and Support Utilities

Sentences

Actuators

Special Interfaces

Tuning
Embedded Coder MISRA C Compliant

After applying Model Advisor MISRA checks

For other violations and releases: See MISRA Compliance Package
Is Code Generation Qualified?

Simulink Code Inspector and Polyspace Qualified under DO-330

Qualified Code Generation with MathWorks Embedded Coder

Natick, Mass. – (21 December 2015)

MathWorks today announced that it has completed the Stage of Involvement 4 (SOI-4) audit for a program involving airborne software for an aircraft system with Transport Canada, the certification authority for the aircraft. As a result of the audit, MathWorks tools and supporting data packages included in the DO Qualification Kit are compliant with DO-330 Tool Qualification Level 4 (TQL-4) for DO-178C and DO-331 certification workflows:

- Qualified model verification
- Qualified code generation and verification
Automating Code Reviews for High Integrity Standards
Simulink Code Inspector
Automate DO-178B/C Code Reviews

Independently verify that Embedded Coder generated code traces to and complies with low-level requirements

- Demonstrate that model and source code match structurally
- Provide model ↔ code traceability data
- Eliminate / reduce manual code reviews for DO-178B/C software
- Same certification credits as qualified code generator
Simulink Code Inspector **Overview**

Model and code development

Independent code inspection

- Static verification tool, that checks the generated code against model
- Automates DO-178B Table A-5/DO-331 Table M.B.A-5 verification activities
Simulink block support

- Most of the Simulink blocks are supported
- Virtual subsystems
- Reference models
- Masked subsystems
- Mask parameters
- Workspace parameters
- Stateflow Objects
- Data Type Replacements
- S-Functions created via Legacy Code Tool*

*Only the function call is verified, not the source code of the S-function
Simulink Code Inspector
User Workflow

1. Compatibility checking

2. Code generation

3. Code inspection

4. Analysis / reporting

- Check model for compatible language subset and code generation settings
- Generate source code
- Verify structural equivalence between model and generated code
- Establish traceability
- Analyse results
- Generate verification evidence
Simulink Code Inspector Advisor

Configure for model reference
- This is the top of the model hierarchy
- Inspect all referenced models

Check model compatibility
- Omit model from code inspection if it fails compatibility checks
  - Check this model

Inspect code
- Generate code before code inspection
  - Report folder: \O178C\D178C_Workflow\Models\slprj\slci
    - Browse...
  - Generate and inspect code

Tips
- To process all enabled items in this folder and generate a new report, click "Run Selected Checks".
- Right-click to select or deselect all items in this folder.
- To automatically display the report after processing, select "Show report after run".
- To display the last report generated, click the "Report" path link.
## Simulink Code Inspector Report for Controller.slx

<table>
<thead>
<tr>
<th>Description</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Inspected Model File</td>
<td>C:\EXPO_2016\PCG\Models\Controller.slx</td>
</tr>
<tr>
<td>Model Version</td>
<td>1.374</td>
</tr>
<tr>
<td>Simulink Version</td>
<td>8.7 (R2016a)</td>
</tr>
<tr>
<td>Checksum when Compiled as Top Model</td>
<td>4182280711 2863547516 1373697292 846543816</td>
</tr>
<tr>
<td>Model Last Modified On</td>
<td>21-Apr-2016 16:28:02</td>
</tr>
<tr>
<td>Inspected Code Files</td>
<td>C:\EXPO_2016\PCG\Models\Controller_ert_rtw\Controller.c</td>
</tr>
<tr>
<td>Inspected Code Files Checksum</td>
<td>22108A9DFA643743C9CCDCAE64C33964</td>
</tr>
<tr>
<td>Code Inspection Run On</td>
<td>21-Apr-2016 16:32:26</td>
</tr>
<tr>
<td>Overall Inspection Result</td>
<td>Passed</td>
</tr>
</tbody>
</table>

**Code Verification Results**: Verified

**Function Interface Verification Results**: Verified
Structural Verification of Generated Code

Summary

- Automates DO-178B Verification Activities
  - Source Code Traces to Model
  - Source Code Complies with Model
- Eliminates / reduces manual code reviews
- Qualifiable as a verification tool
- Implemented independent of Embedded Coder

→ Same benefits as Qualified Code Generator
→ Allows optimized code generation using Embedded Coder
Generating AUTOSAR-compliant code from Simulink® models
What is AUTOSAR?

3-layered Architecture

- Software Architecture Definition
  - Target-independent application development through an RTE and standardized interfaces

- Behavior Modeling & Code Generation

- BSW Configuration & RTE Generation

- Application Layer
  - AUTOSAR Software Component 1
  - AUTOSAR Software Component 2
  - AUTOSAR Software Component n

- Runtime Environment (RTE)
  - Basic Software
    - Services Layer
    - ECU Abstraction Layer
    - Complex Drivers
    - Microcontroller Abstraction Layer

- ECU Hardware
MathWorks AUTOSAR Approach

- **No separate AUTOSAR Blockset needed**
  - Code-generation through Mapping

- **AUTOSAR Software Component Approach with Simulink**
  - Simulink for developing behavior
  - Import and Export of SW Component Description Files (ARXML)

- **Simultaneous generation of C-code and ARXML-Files**
  - Consistency between C-code and ARXML SW-C description files
AUTOSAR Workflows
Top-Down, Bottom-Up & Mixed Approaches
What’s More on AUTOSAR

- From Simulink to Autosar Production Code
- AUTOSAR Code Generation for Multiple Runnable Entities
- AUTOSAR Client-Server SIL Simulation
MathWorks Training Offerings

CODE GENERATION

Fundamentals of Code Generation for Embedded Applications

FUNDAMENTAL
- Simulation speedup with code generation
- Parameter tuning with external mode
- Code generation
- In-the-loop verification
- Code execution profiling

Embedded Coder for Production Code Generation

ADVANCED
- Generated code module and data structure
- Code generation options and optimizations
- Integrating generated code with external code
- Customizing data characteristics
- Advanced customization techniques
- Deploying embedded code

CURRICULUM PATHS

Code Generation/Embedded Systems

Simulink for System and Algorithm Modeling

Fundamentals of Code Generation for Embedded Applications

Embedded Coder for Production Code Generation

http://www.mathworks.com/services/training/