Code Generation Solutions For Embedded Systems

Shobhit Shanker
Senior Application Engineer-Code Generation & Verification
Today’s Agenda

- Challenges For Embedded Software Development
- Embedded Development With Model Based Design
- Typical Tasks For Embedded Code Generation
- Standards Compliance
- Q&A
Challenges For Embedded Software Development

- Presently, development trade-off between:
  - Cost
  - Quality
  - Time

- To break the project development trade-off we have to address:
  - Complexity
  - Optimization
  - Interdependency
  - Verification
  - Tools
The Challenges - Complexity

- Description
  - Increasing functionality
  - Algorithmic, system, and silicon complexity

- Impact
  - Project cost, quality, time

- Solution properties
  - Affordably coping with complexity
  - Scales sub-linearly
The Challenges - Optimization

- Description
  - RAM/ROM Metrics should be within acceptable limits
  - Software should comply with the real time constraints i.e. execution time
  - Design Optimization in reduced time

- Solution
  - It should be possible to estimate these metrics as early as possible
  - The ability to apply code optimization strategies from design/implementation stage
The Challenges - Interdependency

- Description
  - Multiple domains
  - Different parts of the design process are increasingly interdependent
    - Multi-Domain Modeling
    - Behavioral and implementation aspects

- Impact
  - No-cross domain trade-off
  - Multiple “truths”
  - Multiple unconnected Models of Computation

- Solution properties
  - Whole design-flow tools
The Challenges - Verification

- **Description**
  - Verification algorithms do not scale well (state space explosion)

- **Impact**
  - More than half of project development time may be spent on verification

- **Solution properties**
  - Early detection of errors on less detailed models
    - Executable specifications
  - Automated test generation
The Challenges - Tools

- **Description**
  - Embedded software tools excel in aiding low-level development
  - Less support for high-level design
  - Limited integration of embedded software design tools

- **Impact**
  - No seamless support for entire workflow integration
  - Impedes cross-domain optimization

- **Solution properties**
  - High level of interoperability between tools
    - At the model level
    - At a numerical level (e.g., cosimulation)
Solution: Model Based Design Workflow

Executable models
- unambiguous
- only “one truth”

Simulation
- reduces “real” prototypes
- systematic “what-if” analysis

Continuous Test and Verification

Test with Design
- detects errors earlier

Design with Simulation

Automatic Code Generation

Automatic code generation
- minimizes coding errors

Models
Embedded Development Process
with Model-Based Design

- System Requirements
- System Design
- Software Design
- Software Integration
- Coding
- System Integration and Tuning
- Hardware/Software Integration
- Requirements Traceability
- Configuration Management
- Documentation

Sim: Simulation
RP: Rapid Prototyping
OTRP: On-Target Rapid Prototyping
PCG: Production Code Generation
SIL: Software-in-the-Loop Testing
PIL: Processor-in-the-Loop Testing
HIL: Hardware-in-the-Loop Testing
Code Generation: From Design to Implementation

- Embedded Coder (C)
- HDL Coder (Verilog or VHDL Code)
- PLC Coder (Structured Text)
- DSP & µC
- FPGA & ASIC

11
Typical Tasks For Embedded Code Generation

- **Model Preparation**
  - Modeling Guidelines Checking
  - Code Generation Advisor, Code Optimization Settings e.g. RAM/ROM

- **Adding the Software Design Details**
  - Algorithm Partitioning e.g. reusable libraries, Model reference
  - Data Typing, Scoping of Variables, Fixed Point Details

- **Generating the Code**

- **Code Review**
  - Manual review
  - Peer review, Code Walkthrough

- **Numerical Equivalence Testing**
  - Software-in-loop testing (SIL)
  - Processor-in-Loop (PIL) for Target Testing

- **Legacy Code Integration**
Model Preparation—Model Checking, Code Settings
Simulink Verification and Validation

- Static analysis of models against a set of checks
  - Checks for simulation
  - Checks for code generation
  - Requirements Consistency
  - **Modeling Standards**

- Modeling Standards Checks for:
  - MAAB Style Guidelines
  - DO-178B
  - IEC-61508

- Extensibility API
Adding Software Design Details

Data Typing

Subsystem Spec

Simulink Fixed Point

Function and File Partitioning
Fixed-point “translations”

- Determine the optimum data type to reduce over/under-flow, preserve resolution.
- Cast each operand to the optimum data type.
- Perform the operation.
- Cast result to output data type if it is explicitly assigned to be something other than the optimum result data type.
Generating Code

- Hyperlinks
  - Code to Model
  - Model to Code
  - Within Report
Code Review-Simulink Code Inspector

Model and code development

Independent code inspection

- Static verification tool, that checks the generated code against model
- Automates DO-178B Table A-5 verification activities
Integration and “in-the-Loop” Testing

SIL, PIL - Numerical Equivalence Checking
HIL - Integration Testing
Legacy Code Integration

```c
void lct_enum_structparam_step(void)
{
    output1 = foo(&P, &output2);
}
```

Structures and Enums supported

```c
typedef enum {
    ONE = 1,
    TWO = 2,
    THREE = 3
} MyEnum;

typedef struct {
    MyEnum number;
    double value;
} MyStruct;

typedef enum {
    ONE(1),
    TWO(2),
    THREE(3)
} MyEnum;

typedef struct {
    MyEnum number;
    re
```
Software Safety Standards

DO-178B
• Developed for commercial aviation, used elsewhere
• Software Integrity Levels A-E based on hazards
• Level A is highest integrity level, requires MC/DC
• Working on new revision, DO-178C

IEC 61508
• Developed for industrial automation, used elsewhere
• Software Integrity Levels (SIL) 1-4 based on hazards
• SIL 4 is highest possible; SIL 3 is highest practiced

ISO 26262
• Derivative of IEC 61508 for automotive industry
• Currently in draft form

MISRA-AC-AGC®
• Developed for automotive industry, used elsewhere
• More than 100 C programming rules
• Update published in 2004
Verification & Validation of Models and Code
ISO 26262 Example Tool Chain

Textual requirements
Executable specification
Modeling

Simulink, Model coverage (SLVnV), Requirements Management Interface (SLVnV)
Property proving (SLDV)

Module and integration testing at the model level
Reviews and static analysis at the model level

Model Advisor (SLVnV),

Model used for production code generation

Prevention of unintended functionality
Reviews and static analysis at code level

Equivalence testing

Generated code

Object code<sub>Gen</sub>

Code generation

Compilation and linking

PIL testing / CGV (Embedded Coder), Test generation (SLDV)
Traceability report (Embedded Coder), Traceability matrix generation (IEC Cert Kit), Bullseye code coverage integration (Embedded Coder)

Simulink / Stateflow / Simulink Fixed Point

Polyspace

Property proving (SLDV)

Simulink, Model coverage (SLVnV), Requirements Management Interface (SLVnV), Property proving (SLDV)

Requirements Management Interface (SLVnV), Property proving (SLDV)

Module and integration testing at the model level

Reviews and static analysis at the model level

Model Advisor (SLVnV),
Tool Qualification Assessment Results for Embedded Coder

Assessment report
Developing AUTOSAR®-Compliant Software with Simulink® and Real-Time Workshop® Embedded Coder™
Code Metrics

Generated code is smaller than production 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
GM Standardizes on Model-Based Design for Hybrid Powertrain Development

Challenge
Develop new hybrid powertrain technology for GM vehicles

Solution
Standardize on MathWorks tools and Model-Based Design for control systems design and production code generation

Results
- Aggressive delivery date met
- Worldwide collaboration and communication enabled
- Designs reused across product lines

“The Two-Mode Hybrid powertrain took Model-Based Design to a new level within GM. This project provided the confidence and experience we needed to apply MathWorks tools for Model-Based Design on other large-scale global engineering programs.”

Kent Helfrich
General Motors

Link to user story
More than 1.6 million lines of automatically generated flight code certified to DO-178B
HVAC Controller for GM Vehicles Using Model-Based Design

**Challenge**
GM wanted to design an advanced HVAC embedded controller for all GM vehicles worldwide.

**Solution** (with the help of MathWorks’ Consulting)
- Able to use MathWorks tools and Model-Based Design to develop and implement a real-time controller with reusable components.

**Value**
- System models reused across 54 products worldwide.
- Quality improved through early verification.

For more information:
http://www.mathworks.com/tagteam/58943_91713v00_GM_Europe_final.pdf

“Once we had moved to Model-Based Design, we were able to use the same core system in many different vehicles by simply calibrating parameters such as the vehicle dimensions and then re-generating production code” - Johan Hägnander, GM Engineering Europe

GM vehicle dashboard with the HVAC control system installed.
Automatic code generation for production ECU

**Challenge**
Caterpillar wanted to adopt automatic code generation for new algorithms while interfacing to existing development processes, architectures, standards and legacy software

**Solution (with the help of MathWorks’ Consulting)**
- Customized code generation for interfacing to Caterpillar’s legacy processes and software

**Value**
- ECU development effort reduced by a factor of 2 to 4, depending on the project
- ECU development calendar time reduced by a factor of greater than 2

“By adopting automatic code generation and with help from MathWorks Consulting, we have been able to reduce ECU software development time and effort by a factor of 2 to 4, depending on the project.” - Larry E. Kendrick, Caterpillar, Inc.

Source: SAE World Congress : 2004-01-0894 – Caterpillar Automatic Code Generation
Key Takeaways

- Model Based Design using MathWorks tools can be used to address embedded software development challenges
- Code Generation solutions can be used to address common software engineering tasks
  - Software specification, Data definition and Typing, Production Code Generation, Legacy Code Integration, and Report Generation
- MBD workflows exist for addressing industry standards such as DO178, ISO26262, AUTOSAR
- Code Generation Solutions exist for C/C++, HDL, PLC
- Generated code is optimized, readable and comparable to hand-code