Quality Improvement by Continuous Usage of PolySpace with Model-Based Design in the Area of Safety Systems

Dr.-Ing. Almut Hochstädtler
Dipl.-Ing. Jochen Retter

MAC 2008, Stuttgart, 04-June-2008
Contents

- Introduction: How to Establish New Tools
- The Development Process
- Example
  - The Model
  - From the Model to C-Code
  - Checking the C-Code with PolySpace
  - Building an S-Function from the C-Code
  - Testing the S-Function Block
- Summary
How to Establish New Tools

- It is requested in the development guideline...
- The quality department checks its usage!
- It’s the customer’s wish!
- It’s for a single project..

I am convinced and I want to use it, it’s the manager’s wish, the customer is happy and the quality department checks its usage and it is requested in the guideline.

I am convinced and I want to use it.
Introduction, Step by Step Training and Applying

Introduction with Evaluation and Training

Extensive usage and discussion about best practice

2nd training and systematic usage for each module

Continuous improvement by one specialist per customer team

2 months

MathWorks checks with help of PolySpace one code example for each customer group.

Are the developers satisfied?

Agree on best practice and define the process.

Who wants to be a key user?

I am convinced and I want to use it.

It is requested in the development guideline...

Agree on best practice and define the process.

Who wants to be a key user?
Summary Tool Introduction

- Continental Passive Safety uses for a part of the code model-based design and code-generation.
- The robustness of the generated code, the library and the handwritten code are additionally verified via PolySpace.
- The PolySpace run is used prior to the check-in of the single software modules.
- PolySpace is used to prepare code reviews.
- The invest for PolySpace pays off within one [big] project,
  - contributes to reduce the number of V-cycles.
  - reduces the time for code reviews regarding robustness by 75%.
Development Process

- Requirements
- Model in Simulink
- Logically correct model
- C-code
- Semantically correct C-code
- Simulink model based on C-code
- Logically and semantically correct C-code

RTW Embedded Coder
S-Function Builder
PolySpace
Model tests
Requirements

Enable Functionality:
The algorithm shall be executed when the following condition is satisfied:
\[ \text{Percentage} = \text{VALID} \]

Algorithm:
\[ \begin{align*}
\text{RawValue} &= \text{interpolate} \left( \text{Percentage} \right); \\
\text{FilteredValue} &= \left( (\text{FilterSize} - 1) \times \text{ValuePrev} + \text{RawValue} \right) / \text{FilterSize}; \\
\text{ValuePrev} &= \text{FilteredValue}; \\
\end{align*} \]

Input Value:
CAN signal \( \text{Percentage} \).

Output Value:
CAN signal \( \text{FilteredValue} \).

Calibration Values:
\( \text{FilterSize} \): between 0 and 255 with a default value of 25.
Table for \( \text{RawValue} \) depending on \( \text{Percentage} \).

General requirements on calibration values:
Calibration values are specific for every platform and vehicle functionality.
The ECU shall support the capability to program calibrations separately.
Model in Simulink
Model Tests $\Rightarrow$ Logically Correct Model
RTW Embedded Coder ⇒ C-Code

Code Generation Report for ExampleForMAC

Summary

Real-Time Workshop code generated for Simulink model "ExampleForMAC.mdl".

Model Version : 1.311
Real-Time Workshop version : 7.1 (R2008a) 23-Jan-2008
C source code generated on : Fri May 16 14:35:52 2008

Configuration Settings at the Time of Code Generation

[File list and details]
PolySpace

In the Simulink Library there is a block available called “PolySpace for RTW Embedded Coder”.

Paths, directories, file names, and include files are specified automatically.

Options and settings are optimized to analyze auto-generated code.

Export of ranges of global variables from Simulink to PolySpace will be done automatically.

data-range-specifications: 'FilterSize' in [0...255] (init)
data-range-specifications: 'TableData' in [12...88] (init)
data-range-specifications: 'TableIndex' in [0...100] (init)
PolySpace: Division by Zero Check

Code with orange ZDV remark:

```
158  /* Gateway: Filter */
159  /* During: Filter */
160  /* Transition: '<S1>:2' */
161  /* Transition: '<S1>:1' */
162  rtb_Output_1 = (uint8_T)((FilterSize - 1) * Input_2 + rtb_RawValue) / FilterSize;
163  rtb_Output_2 = rtb_RawValue;
```

PolySpace check result description:

```
in "ExampleForMAC.c" line 162 column 71
Source code:
|   rtb_Output_1 = (uint8_T)((FilterSize - 1) * Input_2 + rtb_RawValue) / FilterSize);
|   ^
Warning: scalar division by zero may occur
impossible scalar overflow [/] overflow range: {2147483647 >= [expr]}
```
PolySpace: Overflow Check

Code with orange OVFL remark:

```c
ExampleForMAC.c
157
158 /* Gateway: Filter */
159 /* During: Filter */
160 /* Transition: '<S1>':2' */
161 /* Transition: '<S1>':1' */
162 rtb_Output_1 = (uint8_T)(((FilterSize - 1) * Input_2 + rtb_RawValue) / FilterSize);
163 rtb_Output_2 = rtb_RawValue;
```

PolySpace check result description:

```plaintext
ExampleForMAC.c" line 152 column 17
Source code:
| rtb_Output_1 = (uint8_T)(((FilterSize - 1) * Input_2 + rtb_RawValue) / FilterSize);
|
Warning: scalar variable may overflow on [conversion from unsigned int32 to unsigned int8], range : {[expr] <= ;
scalar variable does not underflow on [conversion from int32 to unsigned int32], range : {[expr] >= 0}
scalar variable does not overflow on [conversion from unsigned int32 to int32], range : {[expr] <= 2147483647}
scalar variable does not overflow on [conversion from unsigned int32 to int32], range : {[expr] <= 2147483647}
scalar variable does not overflow on [conversion from unsigned int32 to int32], range : {[expr] <= 2147483647}
scalar variable does not overflow on [conversion from unsigned int32 to int32], range : {[expr] <= 2147483647}
```
PolySpace ⇒ C-Code with Correct Syntax

- Code:
  - Code that protects against division arithmetic exceptions can be left out.

- Variables:
  - It is checked, if all possible values keep within the limits of their type definition.

- Calibration Values:
  - The complete range of the calibration values is checked.
S-Function Builder ⇒
Simulink Model Based on C-Code

Create an S-Function with the S-Function Builder or the Legacy Code Tool.

You have to define the following things:

- Input Ports: name of input variables.
- Output Ports: name of output variables.
- Data Type Attributes: types of the input and output variables.
- Libraries: directories of all included files; name of the C file; compiler settings defined in defines.txt; *.h files to be included.
- Outputs: name of the function defined in the Real-Time Workshop Embedded Coder interface and the exact source code line for calling the function.

All information can be gathered without searching the C code!
Model Tests ⇒
Logically and Semantically Correct C-Code
Summary

We presented a three-phased procedure to achieve high quality auto-generated software for safety systems:

- logical model check within Simulink®
- semantic check of the auto-generated code with PolySpace
- logical check of the auto-generated code with help of an S-function in Simulink®.

The result of this development process is thoroughly tested C code that is semantically and logically correct.