Production Code Generation (PCG) Time Machine

Tom Erkkinen

Embedded Applications Manager

The MathWorks, Inc.
NACA “Production Coders” - 1949

Advanced Search

find this URL
between these dates
(optional)

First Production Coders

Go Wayback
Wayback Machine
“Now supports two forms of code generation:

- Product enhancements and bug fixes

Enhanced code generation:
- Now supports two forms of optimized code generation: rapid prototyping and production embedded targets
Agenda

PCG Historical Review
- 1990s (Circa Release 11)
- 2000s (Circa Release 14)
- Today (R2008a)
Code Generation – 1990s

**Real-Time Workshop® and Stateflow® Coder**
- Generate code from Simulink and Stateflow that is easy to interact, tune, and experiment with
  - Simulation Acceleration
  - Rapid Prototyping
  - Hardware in Loop (HIL)
- Embedded deployment (some examples)

You can deploy code on any microprocessor using Real-Time Workshop because it generates ANSI-C.
ECU Development – 1990s

Automatic Code Generation usage

- Paper Specs
- Modeling
  - Simulation
  - Acceleration
- Rapid Prototyping
- Coding
- Field Tests & CAL
- HIL Test
- Manual Integration
Code Generation (R11)

Real-Time Workshop® 3.0.1

What's New in Version 3.0.1

- Product enhancements and bug fixes

Enhanced code generation:
- Now supports two forms of optimized code generation: rapid prototyping and production embedded targets

Introduction
Key Features & Benefits
Highlights
Code Generation
Target Environments

MathWorks Automotive Conference '08
Agenda

PCG Historical Review
- 1990s (Circa Release 11)
- 2000s (Circa Release 14)
- Today (R2008a)

New features and trends
- Executable Specification
- Detailed Design
- Code Generation and Integration
- Standards and Certification
ECU Development – 2000s
Using Model-Based Design with Automatic Code Generation

System Specification

Simulation Acceleration

Component Design

Rapid Prototyping

Software Design

On Target Rapid Prototyping

Coding
  • Source

Production Code Generation

System Integration

HIL Test

Component Integration

Processor-in-loop (PIL) Test

Software-in-loop (SIL) Test
Code Generation – Early 2000s

Real-Time Workshop® Embedded Coder
- Supports all Real-Time Workshop options
- Plus generates production code
  - Floating- and fixed-point efficiency
  - Code readability
  - Component integration
  - Golden model
## MAC 2005 - GMPT
### Pilot Result - Metrics

<table>
<thead>
<tr>
<th></th>
<th>Hand Code</th>
<th>Auto-code</th>
<th>Percent Change</th>
</tr>
</thead>
<tbody>
<tr>
<td>Calibration ROM</td>
<td>9464</td>
<td>9464</td>
<td>0%</td>
</tr>
<tr>
<td>Code ROM</td>
<td>2952</td>
<td>2900</td>
<td>-1.76%</td>
</tr>
<tr>
<td>RAM</td>
<td>240</td>
<td>238</td>
<td>-0.83%</td>
</tr>
</tbody>
</table>

More Efficient Than Hand
MAC 2005 - Visteon

Model-Based Development Progress: Overview

Status:
- Visteon Powertrain is using model-based software development globally
- Production intent programs & Advanced projects
- Model-based process: 5 years
- Production intent auto coding: 3 years

Objective:
- Incrementally deploy MBD rather than “big-bang”
  - Model based components must mix with legacy software
- Improve quality while reducing development time and cost
  - Models & simulation improve quality: better understanding leads to better design
  - Reduce development time by employing auto-code generation, test case generation, etc.

Keys to Success:
- Proficiency of users – requires mentoring by experts
- Standardized model style (especially tuned for code generation)
- Visteon custom tools
Model-Based Process - Multi-Target Execution

Goal: One Model, Many Targets

- Single Generic Algorithm Model
- Simulink Data Objects (Data Dictionary)
- Execution on Multiple Targets

Check Readiness for Auto Code

Desktop

Rapid Prototyping Controller

Production Controller

GM
Demo

- Executable specification
  - Multidomain
  - Rapid Simulation
  - Interaction options
- Detailed Design
  - Data Dictionary
- Code generation
  - RP/HIL
  - PCG
  - AUTOSAR
- Verification and Validation
  - Bidirectional trace
  - Model Advisor
Code Generation – Early 2000s

Real-Time Workshop® Embedded Coder
- Generates efficient code that can be customized to look like hand code for
  - Production Code Generation

Embedded Targets
- Provide target specific blocks/features for
  - On-Target Rapid Prototyping
Rapid Prototyping

Simulink

Controller Model

Plant Model

Code Generation

Harness
## Rapid Prototyping Comparison

<table>
<thead>
<tr>
<th></th>
<th>Traditional Bypass</th>
<th>On-Target</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Purpose</strong></td>
<td>New ideas, green field research</td>
<td>Refine designs, production focused</td>
</tr>
<tr>
<td><strong>Production Constraints (efficiency)</strong></td>
<td>From off-the-shelf PC, To custom bypass HW</td>
<td>More</td>
</tr>
<tr>
<td><strong>Hardware (Cost)</strong></td>
<td>From Existing ECUs, To eval board HW</td>
<td></td>
</tr>
</tbody>
</table>
Code Generation – Early to Mid 2000s

Real-Time Workshop® Embedded Coder
- Generates efficient code that can be customized to look like hand code for
  - Production Code Generation

Links and Targets
- Provide target specific blocks/features for
  - On-Target Rapid Prototyping
- SIL and PIL support
MAC 2004 – DC Trucks

Agenda

- Introduction
  - functional and technical overview
- Project description
  - motivation and development process
- C-Code analysis
  - analysing and documentation methods
- Control-module
  - Structure, metrics and co-operation methods
- Auto-Code generation
  - experiences and results
- Testing
  - SIL, PIL and HIL test methods

Conclusions

Results

- Project needs only 18 month until release
  - including analysis, restructuring, modelling and testing
- SIL based function development
  - high state of maturity before vehicle tests start
  - higher test efficiency
  - desktop debugging instead of debugging in vehicle

Code generation

- Embedded Coder meets our demands
  - code efficiency and readability like hand written code
- Project aims could be reached in time!
SIL

Simulink

Controller Model

Plant Model

Host-compiled C, With S-Function Wrapper (DLL)

Code Generation

Options
1. Emulate target word sizes (w/code change)
2. Use target word sizes (w/o code change)
PIL

Simulink

Controller Model

Code Generation

Cross-Compiled C (IDE, ISS)

Plant Model

PIL Options
1. Using ISS
## In-the-Loop Comparison

<table>
<thead>
<tr>
<th>Description</th>
<th>SIL</th>
<th>PIL (ISS)</th>
<th>PIL (HW)</th>
<th>HIL</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Purpose</strong></td>
<td>Verify Source Code Component</td>
<td>Verify Object Code Component</td>
<td>Verify Object Code Component</td>
<td>Verify Complete System Functionality</td>
</tr>
<tr>
<td><strong>Fidelity</strong></td>
<td>Emulated: same source code, not bit accurate&lt;br&gt;Actual: Different source code, bit accurate (fix-pt)</td>
<td>Same object code; bit accurate (fix-pt); not cycle accurate since uses ISS</td>
<td>Same object code; bit accurate (fix-pt); cycle accurate; runs on HW</td>
<td>Same executable code; bit accurate (fix-pt); cycle accurate; emulated I/O</td>
</tr>
<tr>
<td><strong>Platform (Plant, ECU)</strong></td>
<td>Host, Host</td>
<td>Host, Host (ISS)</td>
<td>Host, Target</td>
<td>Real-Time System, Target</td>
</tr>
<tr>
<td><strong>Convenience</strong></td>
<td>Desktop convenient; executes just in Simulink; no HW cost</td>
<td>Desktop convenient; executes on desk or test bench; no HW cost</td>
<td>executes on desk or test bench for processor, ECU, and cables</td>
<td>Executes in test bench or lab; $$ for processor, ECU, I/O, cables</td>
</tr>
<tr>
<td><strong>Real-Time</strong></td>
<td>Non real-time</td>
<td>Real-time (between samples)</td>
<td>Non real-time (between samples)</td>
<td>Hard real-time</td>
</tr>
<tr>
<td><strong>Engineers</strong></td>
<td>Systems or Software Engineers</td>
<td>Software or Test Engineers</td>
<td>Software or Test Engineers</td>
<td>Systems or Test Engineers</td>
</tr>
</tbody>
</table>
Link and Target Products – R2007a

Links
- PIL via IDEs
  - Altium TASKING®
  - Analog Devices VisualDSP++®
  - TI’s Code Composer Studio™
  - Mentor Graphics ModelSim®
  - Cadence® Incisive®
- Project creation
- Optimizations

Targets
- Add-ons to Links
- Device driver blocks
- Optimizations

Many third party offerings
Embedded IDE Link™ MU - R2007b+
(for Green Hills® MULTI®)

- Automates build and test of embedded code using MATLAB®, Simulink® and MULTI® (from Green Hills®)

- For systems and software engineers who want to deploy and verify automatically generated code quickly

- Supports key processors including:
  - Freescale™ MPC5554 and MPC7447
  - NEC V850
  - Analog Devices® Blackfin®
**Code Generation – Mid 2000s**

**Real-Time Workshop Embedded Coder**
- C and C++ Production Code Generation

**Simulink® HDL Coder**
- Verilog and VHDL Code Generation

You can deploy code on any microprocessor or hardware device.
Code Verification – Mid 2000s

Real-Time Workshop Embedded Coder
- C and C++ Production Code Generation

Simulink® HDL Coder
- Verilog and VHDL Code Generation

Links and Targets
- On-Target Prototyping
- SIL and PIL

You can verify code on any microprocessor or hardware device.
Demo

- PIL Testing
Simulation is not the only way to do Verification and Validation
Introduced in 2007...

Simulink Design Verifier
Generate tests and prove model properties using formal methods


Simulink, Stateflow, and Embedded MATLAB Functions
Algorithm and System Design

Generate
Verify

Hand-Generate

C/C++ code
Verilog/VHDL

MCU
DSP
FPGA
ASIC

Delivering safe code - faster
Component Design

System Specification

Simulation Acceleration

Component Integration

System Integration

HIL Test

Rapid Prototyping

Processor-in-loop (PIL) Test

ECU Development – Today (R2008a)

Software Design

Coding

• Source

On Target Rapid Prototyping

Software-in-loop (SIL) Test

Production Code Generation
Agenda

PCG Historical Review
- 1990s (Circa Release 11)
- 2000s (Circa Release 14)
- Today (R2007a to 2008a)

New features and trends
- Executable Specification
- Detailed Design
- Code Generation and Integration
- Standards and Certification
Embedded MATLAB Functions - 2007a

Number of Supported Functions

- **R14**
- **R14sp1**
- **R14sp2**
- **R14sp3**
- **R2006a**
- **R2006b**
- **R2007a**

>>eml_frames, eml_nd
Embedded MATLAB™ Improvements – R2007b

- Generate C using `emlc` in MATLAB®
  - Requires Real-Time Workshop®
- m-files on path supported for Embedded MATLAB in Simulink® and Stateflow®

```matlab
function z = inc(y)
    z = y + 1;
end

% Include files
#include "inc.h"

/* Type Definitions */
/* Variable Declarations */
/* Variable Definitions */
/* Function Declarations */
/* Function Definitions */
real_T inc(real_T eml_y)
{
    return eml_y + 1.0;
}
```
Agenda

PCG Historical Review
- 1990s (Circa Release 11)
- 2000s (Circa Release 14)
- Today (R2008a)

New features
- Executable Specification
- Detailed Design
- Code Generation and Integration
- Standards and Certification
Fixed Point Tool – 2007a

- Includes:
  - Data type override
  - Automated scaling
  - Over/under flow detection
  - Fixed vs. float plots

>>fxpdemo_feedback
Fixed-Point Tool - R2008a

Design Min/Max

Problem
- Fixed-Point Tool scaling based on design min and max data ranges was not supported

Solution
- Fixed-Point Tool now leverages design minimum and maximum from blocks and Stateflow data

Benefit
- Better fits user workflow
- Provides autoscaling that does not require simulation
Fixed-Point Advisor – R2008a
Float- to fixed-point conversion

Problem
- Obtain an initial float- to fixed-point conversion with least effort

Solution
- Fixed-Point Advisor helps
  - Set model parameters
  - Set block parameters
  - Perform fixed-point conversion
  - Validate conversion using floating point results
  - Prepare for code generation
- Complement Fixed-Point Tools, which optimizes fixed-point scaling

>>fxpdemo_fpa
Demo

- Fixed Point Advisor
Agenda

PCG Historical Review
- 1990s (Circa Release 11)
- 2000s (Circa Release 14)
- Today (R2008a)

New features
- Executable Specification
- Detailed Design
- Code Generation and Integration
- Standards and Certification
Code Efficiency - R2007a

Wide Signals (for-loops)

```c
void rtwdemo_forloop_R13SP2_step(void)
{
    real_T rtb_Switch[10];
    int_T il,
    "y0 = rtb_Switch[0];
    ...

int_T il;
    for (il<10, il++)
    { 
        //emo_forloop_R13SP2_U.in[il] * 3.0 >= 0.0
        rtb_Switch[il] = rtwdemo_forloop_R13SP2_U.in[il] - 
                         rtwdemo_forloop_R13SP2_DWork.Delay_DSTATE[il];
    } else {
        rtb_Switch[il] = (rtwdemo_forloop_R13SP2_DWork.Delay_DSTATE[il] -
                         rtwdemo_forloop_R13SP2_U.in[il]) * 5.0;
    }
    rtWk.R13SP2_Y.Outl[il] = rtb_Switch[il];
    rtWk.R13SP2_DWork.Delay_DSTATE[il] = rtb_Switch[il];
}
```

```c
real_T tmp;
    for 
    { 
        ++il;
        if (il < 10, il++)
        {
            rtb_Switch[il] = rtWk.R13SP2_U.in[il] * 3.0;
            rtWk.R13SP2_DWork.Delay_DSTATE[il] = rtb_Switch[il];
        } else {
            tmp = (rtWk.R13SP2_Delay_DSTATE[il] - rtWk.R13SP2_U.in[il]) * 5.0;
            rtWk.R13SP2_Delay_DSTATE[il] = tmp;
        }
    }
```

MathWorks
Automotive Conference '08
Improved Bus Code – R2008a

R2007b

```c
void s_step(void) {
    bus_2 rtb_BusCreator;
    int32_t rtb_size;
    int32_T rtb_signal1;
    uint8_T rtb_p_data[8];
    rtb_size = s_U.in.size;
    (void) memcpy(&(rtb_p_data[0]), &s_U.in.p_data, 8*sizeof(uint8_T));
    s_Y.Out1 = s_U.in.start - s_U.in.end;
    rtb_signal1 = rtb_size << 1U;
    rtb_BusCreator.size = rtb_signal1;
    (void) memcpy(&(rtb_BusCreator.p_data), rtb_p_data, 8*sizeof(uint8_T));
    s_Y.Out4 = rtb_p_data[2];
}
```

R2008a

```c
void s_step(void) {
    s_Y.Out1 = s_U.in.start - s_U.in.end;
    s_Y.Out4 = s_U.in.p_data[2];
}
```

Improved bus code efficiency was a top request for this release
Coder Infrastructure Circa 1999

IR – Intermediate Representation
TLC – Template Language Compiler

TLC-based Interfaces (S-Functions, System Targets (e.g. GRT, ERT))
Coder Infrastructure 2008
- Unified for all domains, highly efficient, extensible

- TLC
- Stateflow
- Embedded MATLAB
- Simscape
- Legacy Code Tool
- Simulink Design Verifier

Multiple IR analyses and optimizations

C, C++, VHDL, Verilog, Makfiles, Projects

Cross product optimizations

IR-based Interfaces (e.g., BuildInfo, Target Function Library, …)

TLC-based Interfaces (e.g., System Targets, Code Templates, …)
Cross Product Optimizations – R2008a

- Optimizations are now enabled across code segments generated for Simulink and Stateflow

**R2007b**

```c
real_T rtb_Gain[10];
...
for (i = 0; i < 10; i++) {
    rtb_Gain[i] = 2.0 * in[i];
}
if (cu_DWork.is_active_c1_cu == 0) {
...
} else {
    for (sf_i1 = 0; sf_i1 < 10; sf_i1++) {
        out[sf_i1] = rtb_Gain[sf_i1];
    }
}
```

**R2008a**

```c
if (cu_DWork.is_active_c1_cu == 0) {
...
} else {
    for (i = 0; i < 10; i++) {
        out[i] = 2.0 * in[i];
    }
}
```
### Example Results

<table>
<thead>
<tr>
<th>Processor</th>
<th>Algorithm Function</th>
<th>Target Specific Generated Code*</th>
<th>Speedup (over ANSI-C)</th>
</tr>
</thead>
</table>
| Tricore® By Infineon® | Fixed-pt Math (int32, int32, saturated) | int32_T tricore_add_s32_s32_s32_sat(int32_T a, int32_T b)  
{  
   return (__sat int)a + b;  
} | 17.1X |
| C6416™ DSP By TI    | FIR 128-sample (int32, 50-taps, saturated) | int32_T TI_C6000_s32_add_s32_s32_s32_sat(int32_T a, int32_T b)  
{  
   return _sadd(a, b);  
} | 6.2X |

*Implemented using Target Function Libraries*
Agenda

PCG Historical Review
- 1990s (Circa Release 11)
- 2000s (Circa Release 14)
- Today (R2008a)

New features
- Executable Specification
- Detailed Design
- Code Generation and Integration
- Standards and Certification
Code Architecture - Integration

Controller Model

Generated Algorithm Code

Legacy Code
Target Code

Scheduler/Operating System and Support Utilities

Comm Drivers
Input Drivers
Output Drivers
Special Device Drivers

Communication Interfaces
Sensors

Actuators
Special Interfaces
Tuning

MathWorks Automotive Conference '08
Subsystem Integration w/Export Functions – R2006a

- Supports a popular scheduling technique in production
- Streamlines code generated
  - No scheduler, no model step function

>> rtwdemo_export_functions
Model Reference – Introduced in Release 14

- Incremental code generation is supported via Model Reference
- When a model is changed, only the dependent model regenerate their code
- Normal mode model reference in R2007b
Model Integration w/Prototype Control – R2008a

Problem
- Previously, non-reentrant code only supported void/void function interfaces

Solution
- Provide function control for top model
  - Pass inports/outports as arguments
  - Pass arguments by value or pointer
  - Control argument names and order
  - No wrapper
- Model Reference (R2008a)

Benefit
- Simplifies code integration and testing
- Reduces global RAM usage

```c
extern int32_T application(int32_T In);
```

Generated step function definition
Pack-and-Go – Introduced in R2006b

- Packages generated code and all dependencies (via zip)
- Includes Referenced Models – R2008a

Code, Pack
(Computer A)

Unpack, Build
(Computer B)
Legacy Code Tool – Introduced in R2006b

- Integrates external code for simulation and code generation (e.g., legacy lookup tables)
- Earlier version of LCT on MATLAB Central

```matlab
>> sldemo_lct_lut
```
## Legacy Code Tool – R2007b

Enhanced for non-computational data integration (device driver, file descriptor,…)

<table>
<thead>
<tr>
<th>Data Type</th>
<th>S-Function IO</th>
<th>S-Function Parameter</th>
<th>S-Function DWork</th>
</tr>
</thead>
<tbody>
<tr>
<td>Built-in data types</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>Simulink.Bus ¹ (scalar only)</td>
<td>Y</td>
<td>N/A</td>
<td>Y</td>
</tr>
<tr>
<td>Simulink.AliasType ¹</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>Simulink.NumericType ²</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>Fixed-point data types ³</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>Fi object</td>
<td>N/A</td>
<td>Y</td>
<td>N/A</td>
</tr>
<tr>
<td>Complex number ⁴</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>Scalar, 1D array, and 2D array</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>ND array</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>Void* &amp; Void **</td>
<td>N</td>
<td>N</td>
<td>** R2007b **</td>
</tr>
</tbody>
</table>
Agenda

PCG Historical Review
- 1990s (Circa Release 11)
- 2000s (Circa Release 14)
- Today (R2008a)

New features
- Executable Specification
- Detailed Design
- Code Generation and Integration
- Certification and Standards
Standards and Certification

- AUTOSAR (previously discussed)
- IEC 61508 Safety Guidelines (discussed tomorrow)
- MAAB Model Guidelines (v2)
Model Standards Checks in Simulink Verification and Validation™

- MAAB v2 checks in R2007b
- IEC 61508 checks in R2008a
MISRA-C Compliance History of Generated Code

- Our MISRA-C test suite consists of several example models
- Results shown for most frequently violated rules

- Improving MISRA-C compliance with each release, such as:
  - Compliant code generated from enabled subsystems (R2007b)
  - Eliminate Stateflow® goto statements (R2007a)
  - Compliant parentheses option available (R2006b)
  - Generate default case for switch-case statements (R2006b)

- MathWorks MISRA-C Compliance Package available upon request
  [www.mathworks.com/support/solutions/data/1-1IFP0W.html](http://www.mathworks.com/support/solutions/data/1-1IFP0W.html)
Stateflow and Embedded MATLAB Traceability – R2008a

Problem

- No automated way to trace Stateflow chart and Embedded MATLAB code to generated code

Solution

- Generate comments that automatically link Stateflow objects and Embedded MATLAB functions to the generated code
- Allow model-to-code and code-to-model bidirectional navigation

Benefit

- Bidirectional traceability helps code reviews, code verification, and software certification
Agenda

PCG Historical Review
- 1990s (Circa Release 11)
- 2000s (Circa Release 14)
- Today (R2008a)

New Features
- Executable Specification
- Detailed Design
- Code Generation and Integration
- Certification and Standards

The Future?
Way Forward Machine
John Deere Production Code Generation - SAE CV 07

1.1 Technical Challenges - Software Complexity is Growing Rapidly

- Overall automotive software complexity is increasing exponentially
- Number of lines of code exponentially growing
- Number of distributed solutions growing rapidly
- Number of systems exponentially growing
- Number of customers exponentially growing

Vehicle Shared Signal Growth

Calendar Year

Number of Shared Signals

Automotive Embedded Software Lines of Code Growth

Lines of Code (Million)

Model Year

20 MLOC in 2013

6.0 Conclusions

- Automotive software systems complexity is increasing exponentially
- Consumer electronics lifecycles will only accelerate this complexity growth
- Model Based design techniques have been proven to significantly improve the quality and robustness of software systems delivery
- Resource constraints & competitive efficiency goals drive the need for risk based prioritisation of scarce resources
- Risk based scaling of Process, Methods and Tools (PMT) enable consistent, traceable and dynamically modifiable PMT allocations.
- Cross domain, cross discipline functionality delivery will increasingly require full lifecycle management disciplines supported by fully integrated enterprise wide PLM systems.
More Info – Product web page

- Supported Hardware
  - All links and targets
  - Incl. Third Parties

- Demos and Webinars
  - Short web clips (2-5 minutes)

- Technical Literature
  - Conference papers

- PCG Eval Kit for Eclipse (Europa)
  - Step by step tutorial (3-4 hours)
  - Earlier version in >>rtwdemos

www.mathworks.com/rtwembedded
More Info – MATLAB Central

Demos
- Lego NXT Robots
- Task Profiling (5554)
- Dual Core (Blackfin)

Tools and Blocksets
- Code Coverage Tool
- Model Assistant Tool
- Target Blocksets

www.mathworks.com/matlabcentral/fileexchange

(Production Code category)
More Info – Support Examples

Categories
- Design Patterns
- Integration with Targets
- Scheduling and Multirate
- Standards and Guidelines

New for 2008a
- Stateflow Pattern Wizard
- Fixed Point Tips for R2008a
- Model Patterns for C constructs
- Best Practices for Date Stores

www.mathworks.com/support/product/examples.html?product=SL&category=all

(Production Code category)
We are very interested in learning about your production code needs.

Thank you!

tom.erkkinen@mathworks.com
One More Thing …

Are you working on or planning for an IEC 61508 project?
Press Room - Latest Press Releases

The MathWorks Real-Time Workshop Embedded Coder Validated by TÜV Rheinland

TÜV Rheinland Industrie Service, ASI Successfully Applied the Automotive Code Validation Suite to Real-Time Workshop Embedded Coder

NATICK, Mass. - (03 Jun, 2008)
The MathWorks today announced that its Real-Time Workshop Embedded Coder product for MathWorks Release R2007b successfully passed the Automotive Code Validation Suite (AVS). AVS provides an independent test suite for validating embedded code generators. According to the report from TÜV Rheinland Industrie Service, ASI, "A successful validation using AVS shall be considered as proof of validation to a recognized procedure according to IEC 61508-3:1998, clause 7.4.4."

AVS lets engineers validate the Model-Based Design tool chain from model specification to final embedded code by assessing the code generator, cross-compiler, linker, and embedded processor. The validation conducted by TÜV Rheinland demonstrated that the results of the C code generated by Real-Time Workshop Embedded Coder matches the Simulink and Stateflow model results for MathWorks Release R2007b. AVS was initiated in 2002 by Continental Automotive Systems, Ford Motor Company, and TÜV Rheinland and has since been opened to automotive companies and tool vendors.

According to Torsten Sauer from Continental Automotive Systems, "This success was a prerequisite for our engineers and project managers to apply Real-Time Workshop Embedded Coder in the development of safety-critical systems; it should encourage everybody to follow this validation approach in order to provide confidence to their specific production configurations."