Production Code Generation
Introduction and New Technologies

Tom Erkkinen
Embedded Applications Manager
The MathWorks, Inc.
Agenda

Historical Review
- Code Generation – 1999 (Release 11)
- Code Generation – 2005 (Release 14)
- Code Generation and Verification – R2007a
Wayback Machine (www.archive.org)
“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
NACA “Computers” - 1949

First Computer Generated Code
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

You can deploy code on any microprocessor using Real-Time Workshop because it generates ANSI-C.
Rapid Prototyping

Simulink

Controller Model

Plant Model

Code Generation

Harness

On-Target Rapid Prototyping Links and Targets
Hardware in Loop (HIL)

Simulink

Controller Model

Plant Model

ECU

Harness

Real-Time Computer

Code Generation
ECU Development – 1990s

Automatic Code Generation usage

Paper Specs

Modeling Simulation Acceleration

Rapid Prototyping

Hand Code

HIL Test

Manual Integration

Field Tests & CAL
Simulink Code Generation - 1999

Release 14

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

- Simulink Control Design
- Simulink Parameter Estimation
Agenda

Historical Review
- Code Generation – 1999 (Release 11)
- Code Generation – 2005 (Release 14)
- Code Generation and Verification – R2007a

New features
- Executable Specification
- Detailed Design
- Code Generation
- Code Integration
ECU Development – 2005
Using Model-Based Design with Automatic Code Generation

System Specification
- Simulation Acceleration

Component Design
- Rapid Prototyping

Software Design
- On Target Rapid Prototyping

Coding
- Source

System Integration
- HIL Test

Component Integration
- Processor-in-loop (PIL) Test

Software Integration
- Software-in-loop (SIL) Test

Production Code Generation
Code Generation – 2005

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

You can deploy code on any microprocessor using Real-Time Workshop Embedded Coder because it generates standard C (ANSI and ISO).
MAC 2005

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
MAC 2005

Goal: One Model for All Platforms

Check Readiness for Auto Code

Generate Code

Data Dictionary
(Switch: Float or Fixed)

Throttle Plate Position Control

Floating-pt Code

Fixed-pt Code
Benefits of Fixed-point Modeling

- Data-type & scaling determination/testing
  - significantly easier and less tedious.
- Data-type mis-match & overflow problems
  - easily detected
- Testing of Fixed point operations
  - Fixed-point computations with much faster iterations
  - Side-by-side testing of fixed-point vs floating-point computation
- Model still looks like model
  - Fixed-point information embedded within Simulink® blocks/data
- Worth doing even with hand-code and no auto code

Benefits of Fixed-point Code Generation

- Significant savings in development time
  - Pilot Project - real work took about 2-3 months (excluding time spent on resolving tools issues)
  - Hand code would have taken at least 6 months
- Fixed-point computation functions/macros
  - Bulk of them already built by Mathworks
- Easier unit testing of generated code
  - Unit testing performed side-by-side with model in MATLAB for comparison
Code Generation – 2005

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

You can deploy code on any microprocessor using Real-Time Workshop Embedded Coder because it generates standard C (ANSI and ISO).
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>Hardware (Cost)</strong></td>
<td>From off-the-shelf PCs To custom bypass HW</td>
<td>From Existing ECUs To eval board HW</td>
</tr>
<tr>
<td><strong>Model Flexibility</strong></td>
<td>More emphasis</td>
<td>Less emphasis</td>
</tr>
<tr>
<td><strong>Code Efficiency</strong></td>
<td>Less emphasis</td>
<td>More emphasis</td>
</tr>
<tr>
<td><strong>Purpose</strong></td>
<td>New ideas, green field research</td>
<td>Refine designs, production focused</td>
</tr>
</tbody>
</table>
MAC 2005

Project Objectives for 2003

• Identify challenges with utilizing automatically generated code from MathWorks Embedded Coder in production controllers
  • Integration of auto-code and hand-code
  • Code / Calibration / RAM placement in memory map
  • Interface to instrumentation tools
  • Maintain code readability and trace ability

• Determine if modifications to GMPT algorithm modeling standards and guidelines were required to support automatic code generation

• Determine if modifications to existing GMPT software process and standards were required to support automatic code generation

• Develop a process and tools to allow code automatically generated from algorithm models to be executed on production controllers for development purposes (On-Target-Rapid-Prototyping)
2003 Project Results

- Some modifications to the GMPT Algorithm Modeling Standards were required to support auto-code
- Some modifications to Software Process / Coding Standards were required to support auto-code
- Efficiency of the auto-code is comparable to hand-code and should not factor into the decision to utilize auto-code
- Auto-code was successfully executed in a production ECU controller
## MAC 2005

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>
MAC 2005

2005 Automatic Code Generation Objectives

- Re-evaluate hand code process and auto-code process to reduce overhead
- Address automatic code generation for multiple large functions in a single model file
- Develop a seamless process for testing the model and the generated code
- Develop on-target rapid prototyping process using Mathworks Embedded Coder
- Improve usability of the automatic code generation process

- Deploy first automatic production code build in actual production powertrain vehicles in Q1 2005
- Rollout production code generation to many additional users and production programs
Real-Time Workshop® Embedded Coder
- Generates efficient code that can be customized to look like hand code for
  - Production Code Generation
- Some SIL support (behavior focused)

Embedded Targets
- Provide target specific blocks/features for
  - On-Target Rapid Prototyping
- Some PIL support (MPC5xx focused)

You can deploy code on any microprocessor using Real-Time Workshop Embedded Coder because it generates standard C (ANSI and ISO).
ECU Development – Today (R2007a)

- **System Specification**
  - Simulation Acceleration
- **Component Design**
  - Rapid Prototyping
- **Software Design**
  - On Target Rapid Prototyping
  - Production Code Generation
- **Component Integration**
  - Processor-in-loop (PIL) Test
- **System Integration**
  - HIL Test
- **Coding • Source**
  - Software-in-loop (SIL) Test

**Simulink Models**
- Executable Specifications from Models
- Continuous Test and Verification
- Design with Simulation
- Implementation with Automatic Code Generation
Agenda

Historical Review
- Code Generation – 1999 (Release 11)
- Code Generation – 2005 (Release 14)
- Code Generation and Verification – R2007a

New features
- Executable Specification
- Detailed Design
- Code Generation
- Code Integration
Code Generation

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

Targets
- Provide target specific blocks and features
  - On-Target Prototyping

You can deploy code on any microprocessor
Code Generation – R2007a

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

Targets
- Provide target specific blocks and features
  - On-Target Prototyping

You can deploy code on any microprocessor
Code Verification – R2007a

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

Targets
- Provide target specific blocks and features
  - On-Target Prototyping
  - PIL

Simulink® HDL Coder
- Verilog and VHDL Code Generation

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

You can verify code on any microprocessor or hardware device.
Code Verification – R2007a

Real-Time Workshop Embedded Coder
- C and C++ Production Code Generation
- SIL Options
  - Emulated or Actual

Target and Link Products
- Provide target specific blocks and features
  - On-Target Prototyping
- PIL Options
  - Incl. many more processors

Simulink® HDL Coder
- Verilog and VHDL Code Generation

You can verify code on any microprocessor or hardware device.
### Portable Word Sizes for SIL – R2007a

#### Problem
- Word sizes for the host and target platforms often differ
- Fixed sizes inhibit software-in-the-loop (SIL) testing of production intent code

#### Solution
- Provide SIL option to size data types during compile (tmwtypes.h on host)
- Only fix data types for deployment (rtwtypes.h on target)

#### Benefit
- Better facilitate desktop verification of production intent code

```c
#ifdef PORTABLE_WORDSIZES
    #include "tmwtypes.h"
#else
    #define __TMWTYPES__
    #include <limits.h>
#endif

typedef signed char int8_T;
typedef unsigned char uint8_T;
typedef short int16_T;
typedef unsigned short uint16_T;
typedef int int32_T;
typedef unsigned int uint32_T;
typedef float real32_T;
typedef double real64_T;
```
SIL

**Simulink**

- Controller Model
- Code Generation
- Host-compiled C With S-Function Wrapper (DLL)

**Plant Model**

*Real-Time Workshop Embedded Coder SIL option*
## SIL vs. HIL Comparison

<table>
<thead>
<tr>
<th>Description</th>
<th>SIL</th>
<th>HIL</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Purpose</strong></td>
<td>Verify Source Code Component</td>
<td>Verify Complete System Functionality</td>
</tr>
<tr>
<td><strong>Platform</strong></td>
<td>Host, Host</td>
<td>Target, Target</td>
</tr>
<tr>
<td>(Plant, ECU)</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Target Fidelity</strong></td>
<td>Emulated: same source code, not bit accurate</td>
<td>Same executable code; bit accurate (fix-pt); cycle accurate; emulated I/O (HIGH)</td>
</tr>
<tr>
<td></td>
<td>Actual: Different source code, bit accurate (fix-pt) (LOW)</td>
<td></td>
</tr>
<tr>
<td><strong>Convenience</strong></td>
<td>Desktop convenient; executes just in Simulink; no HW cost (HIGH)</td>
<td>Executes in test bench or lab; $$ for processor, ECU, I/O, cables (LOW)</td>
</tr>
</tbody>
</table>
PIL – Direct HW
For MPC5xx Microprocessor Hardware

Simulink

Controller Model

Code Generation

Plant Model

Options
1. Direct to HW

Target for Freescale™ MPC5xx
PIL – Indirect HW

For Many Microprocessors Supported by IDE

Options
1. Direct to HW
2. Indirect to HW (IDE) - Today
PIL - ISS

For Many Devices Instruction Set Simulators (ISS)

Options
1. Direct to HW
2. Indirect to HW (IDE) – Today
3. Simulated HW (ISS) - Today
## Processor-In-the-Loop Comparison

<table>
<thead>
<tr>
<th>Description</th>
<th>PIL (ISS)</th>
<th>PIL (HW)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Purpose</strong></td>
<td>Verify Object Code</td>
<td>Verify Object Code</td>
</tr>
<tr>
<td><strong>Platform</strong> (Plant, ECU)</td>
<td>Host, Host</td>
<td>Host, Target</td>
</tr>
<tr>
<td><strong>Fidelity</strong></td>
<td>Same object code; bit accurate (fix-pt); not cycle accurate since ISS (MID)</td>
<td>Same object code; bit accurate (fix-pt); cycle accurate since on HW (HIGH)</td>
</tr>
<tr>
<td><strong>Convenience</strong></td>
<td>Desktop convenient; executes just on host computer w/Simulink and ISS; no HW cost (HIGH)</td>
<td>Executes on desk or test bench; $ for processor board and cables (MID)</td>
</tr>
</tbody>
</table>

*ISS Instruction Set Simulator*
# 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>Emulated: same source code, not bit accurate</td>
<td>Actual: Different source code, bit accurate (fix-pt)</td>
<td>Same source code, bit accurate (fix-pt); not cycle accurate unless ISS</td>
<td>Same executable code; bit accurate (fix-pt); cycle accurate; emulated I/O</td>
</tr>
<tr>
<td><strong>Fidelity</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>Platform (Plant, ECU)</strong></td>
<td>Desktop convenient; executes just in Simulink; no HW cost</td>
<td>Desktop convenient; executes on host computer with Simulink and HW cost</td>
<td>Executes on desk or test bench for processor and cables</td>
<td>Executes in test bench or lab; $$ for processor, ECU, I/O, cables</td>
</tr>
<tr>
<td><strong>Convenience</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>Real-Time</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>
<tr>
<td><strong>Engineers</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
MAC - 2004

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!
Link and Target Products

Links
- PIL via Integrated Development Environments (IDEs)
  - Altium TASKING®
  - Analog Devices VisualDSP++®
  - TI Code Composer Studio™
  - Mentor Graphics ModelSim®
  - Cadence® Incisive®
- Project creation for prototyping or production

Targets
- Work on top of Links (are add-ons)
- Device driver block sets
- Processor specific optimizations
Links and Targets Product Providers

Availability

- MathWorks
  - Products
  - Consulting
- MATLAB Central
- Third Parties
<table>
<thead>
<tr>
<th>Title</th>
<th>Submitted</th>
<th>Rating</th>
<th>Author</th>
<th>Category</th>
<th>Score</th>
<th>Reviews</th>
</tr>
</thead>
<tbody>
<tr>
<td>Embedded Coder Robot NXT Demo</td>
<td>2006-12-15</td>
<td>4.25</td>
<td>Tom Ekholm</td>
<td>Production Code</td>
<td></td>
<td></td>
</tr>
<tr>
<td>TLiC Quick Start Guide</td>
<td>2006-04-18</td>
<td>4.0</td>
<td>John Sjodin</td>
<td>Production Code</td>
<td></td>
<td></td>
</tr>
<tr>
<td>LPC55S Motor Control Function Block</td>
<td>2006-03-20</td>
<td>4.0</td>
<td>Eric Denton</td>
<td>Production Code</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Address Specific Parameter Custom Storage Class MemMix</td>
<td>2007-05-14</td>
<td>0</td>
<td>Pete Szpul</td>
<td>Production Code</td>
<td></td>
<td>0</td>
</tr>
<tr>
<td>Using MATLAB tools to apply Model-Based Design for COM/17B Applications</td>
<td>2007-03-20</td>
<td>5.0</td>
<td>Vinod Cheriyan</td>
<td>Production Code</td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>Integrating Processor-Specific Code with Model-Based Design of Embedded Systems</td>
<td>2007-02-12</td>
<td>0</td>
<td>Tom Ekholm</td>
<td>Production Code</td>
<td></td>
<td>0</td>
</tr>
<tr>
<td>External I/O and State Information Block</td>
<td>2006-07-19</td>
<td>5.0</td>
<td>Roger Aarestad</td>
<td>Production Code</td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>Processor In the Loop with Link for TASKING</td>
<td>2007-03-09</td>
<td>0</td>
<td>David MacIay</td>
<td>Production Code</td>
<td></td>
<td>0</td>
</tr>
<tr>
<td>Optimized Infineon TriCore Simulink Blocks for use with Link for TASKING</td>
<td>2007-02-23</td>
<td>0</td>
<td>John Sjodin</td>
<td>Production Code</td>
<td></td>
<td>0</td>
</tr>
<tr>
<td>Model Assistant Tool</td>
<td>2002-12-06</td>
<td>2.0</td>
<td>Pete Szpul</td>
<td>Production Code</td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>Code Coverage Tool</td>
<td>2007-03-15</td>
<td>3.29</td>
<td>Mark Walker</td>
<td>Production Code</td>
<td></td>
<td>7</td>
</tr>
<tr>
<td>C-CAN drivers for use with Target for Infineon C166</td>
<td>2007-03-09</td>
<td>0</td>
<td>David MacIay</td>
<td>Production Code</td>
<td></td>
<td>0</td>
</tr>
</tbody>
</table>
# Third Party Links and Targets

<table>
<thead>
<tr>
<th>Processors</th>
<th>C/C++ Support</th>
<th>Built-In Data Type Support</th>
<th>Link Products</th>
<th>Target Products</th>
<th>Third-Party Products</th>
</tr>
</thead>
<tbody>
<tr>
<td>All processors</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>32-bit PowerPC</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8051 Compatible</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>AMD K5/K6/Athlon</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ADI Blackfin</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ADI SHARC</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ARM 7/9/9</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Freescale MPC86xx,</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MPC7xx, and</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MPC52xx (32-bit</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Freescale Motrix</td>
</tr>
<tr>
<td>PowerPC)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Freescale MPC55xx</td>
<td></td>
<td></td>
<td></td>
<td>Freescale Motrix</td>
<td></td>
</tr>
<tr>
<td>Freescale MPC5xx</td>
<td></td>
<td></td>
<td>Target for</td>
<td>Various</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Freescale</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>MPC5xx</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Select the manufacturer to see a full list of supported hardware.

www.mathworks.com/products/rtwembedded/supportedio.html
Agenda

Historical Review
- Code Generation – 1999 (Release 11)
- Code Generation – 2005 (Release 14)
- Code Generation and Verification – R2007a

New features
- Executable Specification
- Detailed Design
- Code Generation
- Code Integration
Embedded MATLAB Functions - 2007a

Number of Supported Functions

<table>
<thead>
<tr>
<th></th>
<th>R14</th>
<th>R14sp1</th>
<th>R14sp2</th>
<th>R14sp3</th>
<th>R2006a</th>
<th>R2006b</th>
<th>R2007a</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fi</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Non-Fi</td>
<td>50</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

>>eml_frames, eml_nd
Agenda

Historical Review
- Code Generation – 1999 (Release 11)
- Code Generation – 2005 (Release 14)
- Code Generation and Verification – R2007a

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

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

>>fxpdemo_feedback
Agenda

Historical Review
- Code Generation – 1999 (Release 11)
- Code Generation – 2005 (Release 14)
- Code Generation and Verification – R2007a

New features
- Executable Specification
- Detailed Design
- Code Generation
- Code Integration
Wide Signals (for-loops)

```c
void rtwdemo_forloop_R14SP2 ( void )
{
    for ( int_T ii;
         ii < 10; ii++)
    {
        real_T rtb_Switch[10];
        rtb_Switch[0] = 0.0;
    
        for ( int_T il;
             il < 10; il++)
        {
            real_T rtb_Switch[10];
            rtb_Switch[0] = 0.0;
        
            for ( int_T ii;
                 ii < 10; ii++)
            {
                real_T rtb_Switch[10];
                rtb_Switch[0] = 0.0;
            }
        }
    }
}
```
MISRA-C 2004 Support – R2007a

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

- Improving MISRA-C compliance with each release, e.g.
  - 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
  http://www.mathworks.com/support/solutions/data/1-1IFP0W.html
Agenda

Historical Review
- Code Generation – 1999 (Release 11)
- Code Generation – 2005 (Release 14)
- Code Generation and Verification – R2007a

New features
- Executable Specification
- Detailed Design
- Code Generation
- Code Integration
Model Integration via Model Reference – R14

- Incremental code generation is supported via Model Reference

Model dependencies:

```
% Specify the model dependencies as a cell array of the form
% automatically include the model, mdl and linked lib/yn, n not on the MATLAB path, use absolute path, prefix $:
% path is relative to the location of the mdl file, wildcard % to comment out a line, use '...' to continue lines. For example:
% {'%D\Work\parameters.mat', '$MDL\mdlvars.mat', ...
% {'D:\Work\mylib\util\lib.m'}}
```
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 Integration w/Function Prototypes - R2007a

Function control for top model
- Pass inports/outports as arguments
- Pass arguments by value or pointer
- Control argument names and order

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

Generated step function definition
### Production Code Integration Comparison

<table>
<thead>
<tr>
<th></th>
<th>Model Reference Integration</th>
<th>Model Step Function Integration</th>
<th>Subsystem (Function Export) Integration</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Fidelity / Control</strong></td>
<td>LOW-MID</td>
<td>MID</td>
<td>HIGH</td>
</tr>
<tr>
<td><strong>Convenience</strong></td>
<td>Fully automated, incremental code HIGH</td>
<td>Partial function control</td>
<td>Manual integration LOW</td>
</tr>
<tr>
<td><strong>Applications</strong></td>
<td>Large Scale Model, System Focused</td>
<td>Model Bound, System or Software</td>
<td>Software Engineering Focused</td>
</tr>
</tbody>
</table>

- Packages generated code and all static dependencies (via zip)
- Built on BuildInfo API
Code Architecture - Integration

- Communication Interfaces
- Sensors
- Input Drivers
- Comm Drivers
- Generated Algorithm Code
- Included Target Optimized Code
- Scheduler/Operating System and Support Utilities
- Output Drivers
- Special Device Drivers
- Tuning
- Special Interfaces
- Actuators
Legacy Code Tool – R2006b

- Integrates external code for simulation and code generation (e.g., legacy lookup tables)

>> sldemo_lct_lut
Fixed-Point Advanced Lookup Tables – R2006b

- Two blocks
  - Prelookup
  - Interpolation using prelookup
- Support
  - Floating- and fixed-point
  - n-D support
- Optimized
  - Share prelookup costs
  - Saturation-free
  - Allows optimal data types
  - Sub-table selection
  - Cross block range check
Agenda

Historical Review
- Code Generation – 1999 (Release 11)
- Code Generation – 2005 (Release 14)
- Code Generation and Verification – R2007a

New Features
- Executable Specification
- Detailed Design
- Code Generation
- Code Integration

The Future?
Way Forward Machine

Internet Archive Wayback Machine - Microsoft Internet Explorer provided by The Mathworks, Inc.

Advanced Search

find this URL

between these dates (optional)

R2007b?
Way Forward Machine

Application Generation

Scheduler/Operating System and Support Utilities

Advanced Search

find this URL
between these dates
(optional)

Production Code Generation?
Way Forward Machine

Ask the experts.

Advanced Search

find this URL
between these dates
(optional)

Production Code Growth?
Exponential Growth of Embedded Code

Estimated Source Lines of Code (LOC)

- Today’s powertrain: 500,000 LOC
- Today’s vehicles: 1,000,000 LOC
- 2015 vehicles: 100,000,000 LOC

Managing for software success

The automotive industry is providing ever "smarter" vehicles. Intelligence in the car now does everything from ensuring a comfortable temperature zone independently for each occupant to enabling the creation of a whole new class of vehicles called hybrids. As embedded control functionality grows, managing the risks of the software development process requires robust management strategies.

A lot of functionality means a lot of software. Current estimates for the software load in a vehicle range from 500,000 source lines of code (SLOC) for powertrain functions alone to over 1,000,000 SLOC if all functionality is included.

"The growth of the top end of automotive embedded software, as for the PC world, has generally been exponential," said Robert Gee, Director of Strategy for Systems and Software within Motorola’s Automotive Business. "Some automotive devices alone today may contain several million lines of code. These devices have been and will likely continue to be integrated into the vehicle in volume. It is becoming more integrated than ever before. Functions that were once developed separately, such as powertrain and vehicle stability controls, now interact with each other. The trend toward standardization of bus architectures instead of point-to-point wiring also increases the potential for software integration issues."

Robert Gee
Director of Strategy for Motorola Automotive

“Growth of top end automotive embedded software has been exponential.”

Automotive Engineering, “Managing for Software Success” – Aug 2006
sae.org/automag/electronics/08-2006/1-114-8-34.pdf
Explosive Growth of Code

1.1 Technical Challenges - Software Complexity is Growing Rapidly

- Overall automotive software complexity is growing exponentially
  - Number of Lines of Code is growing exponentially
  - Number of distributed software-system solutions growing rapidly
  - Number of system dependencies (coupling) growing
  - Number of Customer input signals growing exponentially

20 MLOC in 2013

Software System Engineering with Model-Based Design,
International Conference on Software Engineer, ICSE May 2007
Christopher Davey, Ford Motor Co and Jon Friedman The MathWorks, Inc
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.
Conclusions

Jorgen Hansson of the Software Engineering Institute advocates the use of modeling as a way to reducing the amount of testing required.

GM’s Jim Kolhoff confirms that strategy.
More Info – Later Today

- Papers
  - Scania, Ford, …
- Master Classes
  - Knock Detection
- Exhibits and Demos
  - Production Code for Body Applications
  - Freescale, Mototron, …

1:30 – 3:00 p.m.

Master Class: Knock Detection Algorithm: From Design to Hardware/Software Implementation via HDL or C

Mark Corless,
The MathWorks

Prashant Rao,
The MathWorks
More Info – Next Week

PCG Workshop
- Full Day
- Hands-On
- Free

Details
- Date: Tuesday, June 26, 2007
- Time: 8:30 a.m. – 4:45 p.m.
- Location: The MathWorks Novi, MI

Request at
- www.mathworks.com/seminars/ecunovi

Workshop On ECU Production Code Generation Using Simulink®

Workshop Highlights
Through automotive ECU application examples, including projects developed for resource-constrained microprocessors.
- Establish a software engineering environment that focuses on developing a detailed software model.
- Generate and optimize ANSI C code for memory-constrained microprocessors.
- Produce clearly commented, well-partitioned, and traceable code.

The following topics on software architecture, design, and integration will be covered in detail:
- Model referencing and bus objects for component-based development
- Model explorer to establish and manage a data dictionary
- Data definition and data types selection
- Data objects and alias types to create and import complex data definitions
- Module packaging features to comment, partition, and package code
- Simulink and Stateflow to integrate legacy code for simulation and code generation

PCG Workshop
More Info – Next Month

Live Webinar
- Live Q&A

Details
- Date: July 17, 2007
- Time: Varies
- Duration: 1 hour

Request at
- www.mathworks.com/webinars

Event Status

This event has not started.

Enroll  Refresh  Go Back

Event Information

Event: Embedded Code Generation & Verification Using Simulink

Date and Time: Tuesday, July 17, 2007 9:00 am
Eastern Daylight Time (GMT -04:00, New York)  Change time zone
  July 17, 2007 2:00 pm
  GMT Daylight Time (GMT +01:00, London)
  July 17, 2007 9:00 pm
  China Standard Time (GMT +08:00, Beijing)
  July 17, 2007 11:00 pm
  Australia Eastern Standard Time (GMT +10:00, Sydney)

Panelist(s) Info: Tom Erkkinen

Duration: 1 hour

Description: This webinar starts with a review of how Real-Time Workshop Embedded Coder generates high-quality and efficient C code from Simulink and Stateflow models for deployment on embedded systems. It then shows how the models and generated code can be analyzed and tested using processor-in-the-loop cosimulation.

The webinar includes new features important for Model-Based Design involving:

- Algorithm design
- Data management
- Software integration
- Requirements traceability

This webinar is for people familiar with Simulink but not experienced with automatic code generation and verification. Although, experienced code generation users using old releases may be interested in some of the new features and products presented.

A Q&A session will follow the presentation.
More Info – Anytime

PCG Guided Tutorial
- 3-4 hours
- Detailed Intro

Request at
- www.mathworks.com/rtwembedded
- Production Code Generation Eval Kit

Module 0: Introduction

The exact process for designing and implementing a control algorithm varies from one organization to the next. However, some basic steps in the process are common. This tutorial provides an interactive experience of applying MathWorks products to those common basic steps. You work with a supplied Simulink model and using Real-Time Workshop Embedded Coder, generate code for the model, integrate the generated code with an existing system, and validate simulation and executable results.

Contents

- Format
- Prerequisite Knowledge
- Using This Tutorial
- Navigation

Format

This tutorial consists of 8 modules, including this introduction. The modules are listed here and at the end of the module in the Navigation section.

- Module 0: Introduction
- Module 1: Understanding the Model
- Module 2: Configuring the Data Interface
- Module 3: Function Partitioning within the Generated Code
- Module 4: Calling External C Code from the Simulink Model and Generated Code
- Module 5: Integrating the Generated Code into the External Environment
- Module 6: Testing the Generated Code
- Module 7: Evaluating the Generated Code
Next Steps – Anytime

Contact Us:
- www.mathworks.com/aboutus/contact_us

We are very interested in learning about your production code needs.

Thank you – tom.erkkinen@mathworks.com
One More Thing …
Way Forward Machine (www.archive.org)

Software Engineers? Certainly Human!

Advanced Search

find this URL between these dates (optional)
Add Links to Requirements

Requirements appear in the code

```matlab
/* DiscretePulseGenerator: '<Root>/clock'

* Requirements for '<Root>/clock':
* 1. Clock period shall be consistent with chirp tolerance
*/
rtb_clock =
(rtDWork.clockTickCounter < 1.0 &&
 rtDWork.clockTickCounter >= 0) ?
1.0 :
0.0;
if (rtDWork.clockTickCounter >= 2.0-1) {

>> rtwdemo_requirements
```
Enforce Modeling Standards

Simulink Model Advisor
- Select from built-in checks
- Add customized check using API
- Archive report as evidence
Compliance history of library files

- Library files which can be used in production code are
  - Provided in rtw/c/libs/src dir
  - auto generated in _sharedutils dir
- In R2007a, complete compliance for
  - Declarations (Rules 8.1 through 8.12)
  - Control flow (Rules 14.1 through 14.10)
  - Functions (Rules 16.1 through 16.10)
  - Standard libraries (Rules 20.1 through 20.12)
- Improving compliance with each release for
  - Identifiers
  - Types
  - Arithmetic & Pointer Conversions
  - Expressions
  - Pointers and Arrays
  - Pre-processing directives
# Documentation of the Generated Code

<table>
<thead>
<tr>
<th></th>
<th>HTML report</th>
<th>Code node</th>
<th>Pre-canned report</th>
<th>Report generator</th>
</tr>
</thead>
<tbody>
<tr>
<td>Need setup</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>Customizable</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>Link to model objects</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>Portable document</td>
<td>Yes*</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Containing model snapshots</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Multiple file formats</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>Common use case</td>
<td>Interactive review</td>
<td>Interactive review</td>
<td>Quick deposition</td>
<td>Formal documentation</td>
</tr>
<tr>
<td>Require Report Generator</td>
<td>No</td>
<td>No</td>
<td>Installation Required</td>
<td>Installation and license Required</td>
</tr>
</tbody>
</table>
Independent, Integrated Code Reports

- Standard HTML report of generated code using Real-Time Workshop
- Customize reports of model and generated code using Simulink Report Generator
  - HTML, PDF, RTF, DOC
- Standalone: no license required to view previously generated reports

```
>> rtwdemo_codegenrpt
>> help rtwReport
```
Creating Customized Report

Pre-made components

Customized components

sources

setup files

.stylesheet

.rgs

.rpt

.xml

.html

.pdf

.doc

.customized reports
RTW BuildInfo API

- **Problem Statement**
  - What are the dependencies and artifacts of the generated code?

- **Requirements**
  - Provide a list of all dependencies for the generated code
  - Enable easier integration with third-party tools, source control systems, and IDEs.

- **Solution**
  - An M-based API that contains
    - All source files and their fully qualified paths
    - All header files and their fully qualified paths
    - Compiler/linker options, build options
  - Accessible
    - During an RTW build process
    - From a .mat file for post-build access

rtwdemo_buildinfo
Building Info API (Example)

```matlab
function buildinfo_demo(buildInfo)

    % Write all generated files to an HTML report
    fid = fopen('BuildInfoReport.html', 'w+');
    fprintf(fid, '<html>
    fprintf(fid, '<title>Build Info API</title>
    fprintf(fid, '<h1>Build Info API Demo</h1>

    % Get list of generated source files
    allSrcFileList = buildInfo.getFiles('all', false, false);

    fprintf(fid, '<h4>Generated Source and Header Files:</h4>
    fprintf(fid, '%<br>', allSrcFileList(i));
    end

    fprintf(fid, '</html>
    fclose(fid);
    web(''BuildInfoReport.html'');
end
```

```
>> load_system('rtwdemo_counter')
>> set_param(bdroot,'PostCodeGenCommand','buildinfo_demo(buildInfo)')
>> rtwbuild('rtwdemo_counter')
```
Support for Project-Based IDEs

- Optionally generate a makefile
- Template makefile only required when ‘Generate makefile’ is selected
- Ideal for interfacing with Project-Based IDEs
Shared Library Target Solution

Problem

- Better support for host-based system-level simulators
- Multiple application programming environments (C/C++, Fortran, MS Visual Basic, Borland Delphi, Java)

Solution

- RTW-EC shared library target
- Generate a shared library (.dll or .so) version of the generated code with documented interface
- Support initialize, step, and terminate a simulation
- Unix and PC support
Shared Library Target Benefits

Benefit

- Easier distribution of model simulation
- Shared library is **reusable** and **upgradeable** without recompiling the overall application
- Package and secure **intellectual property**
Configuring a Model Reference Hierarchy

- Before R2007a:
  - Always re-copy all configuration sets

Problem:
- Hard to manage - Multiple copies
- All referenced models must be modified
  - Must unlock file for writing
- No easy way to use different configurations/targets
Configuring a Model Reference Hierarchy

- **Workflow Solution: full configset reference**
  1. Add a `ConfigSet` in the base workspace
  2. Add a `ConfigSetRef` to each model
- Change configuration in one place
  - Reuse models in other applications
  - Reconfigure target in one step
  - Keep models locked

```matlab
>> cs = Simulink.ConfigSet
or
>> cs = getActiveConfigSet(mdl);
```

```
>> mdlref_configset_ref
```
Model Dependencies

- Model Dependency Viewer (R2006b)
  - Easy to understand high level design
- Model Dependencies Tools (R2007a)
  - Lists files required by your model in a “manifest” file
  - Packages the model with required files into a zip file
  - Compare older and newer manifests for the same model
  - Captures .mdl, .mat, .m files dependencies
- Easy to see component dependencies
  - Detects changes in lower level components that impacts components higher in the hierarchy
Model File Change Notification

- Problem: when using Simulink with revision control,
  - Open a Simulink model, x.mdl
  - Retrieve a new version of file x.mdl from revision control system
  - When users Simulate, edit, generate code using x, which version are they using?
- Solution in R2007a:
  - Model File Change Notification
    - See Simulink preferences