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)
Simulink Code Generation - 1999

“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

Advanced Search

find this URL
between these dates
(optional)
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
Hardware in Loop (HIL)

Simulink

Controller Model

Plant Model

Code Generation

Harness

ECU

Real-Time Computer
ECU Development – 1990s

Automatic Code Generation usage

Hand Code

- Manual Integration
- HIL Test
- Field Tests & CAL

- Rapid Prototyping
- Simulation Acceleration
- Modeling

Paper Specs
Simulink Code Generation - 1999

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

Featured Products:
- Bioinformatics Toolbox
- Instrument Control Toolbox
- Data Acquisition Toolbox
- Image Acquisition Toolbox
- Link for ModelSim®
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
Production Code Generation

System Integration
HIL Test
Component Integration
Processor-in-loop (PIL) Test
Software Integration
Software-in-loop (SIL) Test

Using Model-Based Design with Automatic Code Generation

ECU Development – 2005
The MathWorks
MATLAB® & SIMULINK®
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
Goal: One Model for All Platforms
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)
MAC 2005

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>
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
Code Generation – 2005

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
- Component Design
- Software Design
- Coding • Source
- Software Integration
- System Integration

- Simulation Acceleration
- Rapid Prototyping
- On Target Rapid Prototyping
- Production Code Generation
- Processor-in-loop (PIL) Test
- Processor-in-loop (SIL) Test
- HIL Test

Using Model-Based Design 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

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

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

Deployment

```
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

Plant Model

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

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><strong>(Plant, ECU)</strong></td>
<td><strong>Target</strong> &lt;br&gt;Emulated: same source code, not bit accurate &lt;br&gt;Actual: Different source code, bit accurate (fix-pt) (LOW)</td>
<td>Same executable code; bit accurate (fix-pt); cycle accurate; emulated I/O (HIGH)</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

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)

- Controller Model
- Plant Model
- Cross-Compiled C Instruction Set Simulator (ISS)
- IDE Link

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></td>
<td>Host, Host</td>
<td>Host, Target</td>
</tr>
<tr>
<td><strong>(Plant, ECU)</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>Fidelity</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>
<tr>
<td><strong>Convenience</strong></td>
<td><em>ISS Instruction Set Simulator</em></td>
<td></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>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</td>
<td>Partial: object code; bit accurate (fix-pt); not cycle accurate; uses ISS</td>
<td>Same: object code; bit accurate; cycles ISS</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 just in Simulink and HW cost</td>
<td>Same: 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>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>
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

Simulink®

TASKING

ST10

Tricore
<table>
<thead>
<tr>
<th>Title</th>
<th>Description</th>
<th>Submitted</th>
<th>Rating</th>
<th>Category</th>
<th>Author</th>
<th>Rating</th>
<th>Reviews</th>
</tr>
</thead>
<tbody>
<tr>
<td>External I/O and State Information Block</td>
<td>This block generates an extra file during code generation containing external inputs, outputs an Author: Roger Aamodt</td>
<td>2006-07-19</td>
<td>5.0</td>
<td>Production Code</td>
<td>Roger Aamodt</td>
<td>1 review</td>
<td></td>
</tr>
<tr>
<td>Processor in the Loop with Link for TASKING</td>
<td>This demo shows how Processor in the Loop (PIL) testing can be used for verification of automatically Author: David Maclay</td>
<td>2007-03-09</td>
<td>0</td>
<td>Production Code</td>
<td>David Maclay</td>
<td>0 reviews</td>
<td></td>
</tr>
<tr>
<td>Optimized Infineon TriCore Simulink Blocks for use with Link for TASKING</td>
<td>Simulink blocks (FIR, FFT) optimized for the Infineon TriCore using the Infineon TriLib DSP library.</td>
<td>2007-02-23</td>
<td>0</td>
<td>Production Code</td>
<td>John Tiddeman</td>
<td>0 reviews</td>
<td></td>
</tr>
<tr>
<td>Model Assistant Tool</td>
<td>Using the Model Assistant Tool, you can quickly configure a model for code generation.</td>
<td>2002-12-06</td>
<td>2.0</td>
<td>Production Code</td>
<td>Pete Szpak</td>
<td>1 review</td>
<td></td>
</tr>
<tr>
<td>C-CAN drivers for use with Target for Infineon C166</td>
<td>Support for C-CAN Hardware used on newer variants of the ST10 microcontroller Author: David Maclay</td>
<td>2007-03-09</td>
<td>0</td>
<td>Production Code</td>
<td>David Maclay</td>
<td>0 reviews</td>
<td></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, MPC7xx, and MPC52xx (32-bit PowerPC)</td>
<td>✔</td>
<td></td>
<td></td>
<td></td>
<td>Freescale Motorcon</td>
</tr>
<tr>
<td>Freescale MPC5xx</td>
<td>✔</td>
<td></td>
<td></td>
<td></td>
<td>Freescale Motorcon</td>
</tr>
<tr>
<td>Freescale MPC5xx</td>
<td>✔</td>
<td></td>
<td></td>
<td></td>
<td>Freescale Motorcon</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

```
>> eml_frames, eml_nd
```

## Number of Supported Functions

- **Fi**
- **Non-Fi**

### MATLAB Release
- R14
- R14sp1
- R14sp2
- R14sp3
- R2006a
- R2006b
- R2007a
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

```plaintext
>> 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_R13SP2_step(void)
{
    real_T rtb_Switch[10];

    for (int_T i = 0; i < 10; i++)
    {
        y0 = rtb_Switch[i];
    }
}
```

```c
void rtwdemo_forloop_R14SP2_step(void)
{
    real_T rtb_Switch[10];

    for (int_T i = 0; i < 10; i++)
    {
        y0 = rtb_Switch[i];
    }
}
```

```c
void rtwdemo_forloop_R2007a_step(void)
{
    real_T tmp;

    for
    
    ++i { (i) >= 0.0) {
        y0 = rtb_Switch[i];
    } else {
        tmp = (rtb_Switch[i] - rtb_Switch[i]) * 5.0;
    }
}
```

```c
const real_T *u0 = &rtb_Switch[0],
real_T *dw_DSTATE = &rtwdemo_forloop_R13SP2_DWork.Delay_DSTATE[0];

for 0; i < 10; i++)
```

```c
    y0 = rtb_Switch[i];
    rtb_Switch[i] = tmp;
    rtb_Switch[i] = tmp;
}
```
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
Code Architecture - Integration

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

Controller Model
Model Integration via Model Reference – R14

- Incremental code generation is supported via Model Reference

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

```c
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>Prototype control</td>
<td>Manual integration, LOW</td>
</tr>
<tr>
<td><strong>Applications</strong></td>
<td>Large Scale Models, System Focused</td>
<td>Model Boundaries, 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

Controller Model

Generated Algorithm Code

Included Target Optimized Code

Input Drivers

Comm Drivers

Output Drivers

Special Device Drivers

Special Interfaces

Scheduler/Operating System and Support Utilities

Tuning

Actuators

Communication Interfaces

Sensors
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
Way Forward Machine

Advanced Search

find this URL
between these dates
(optional)

Scheduler/Operating System and Support Utilities

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

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

Robert Gee
Director of Strategy for Motorola Automotive

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
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
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
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 Erkkilä
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!
Coders?
Add Links to Requirements

Requirements appear in the code

```matlab
94 /* DiscretePulseGenerator: '<Root>/clock'
95 *
96 * Requirements for '<Root>/clock':
97 * 1. Clock period shall be consistent with chirp tolerance
98 */
99
100 rtb_clock =
101 (rtDWork.clockTickCounter < 1.0 &&
102 rtDWork.clockTickCounter >= 0) ?
103 1.0 :
104 0.0;
105 if (rtDWork.clockTickCounter >= 2.0-1) {
```
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/libsfc 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>Feature</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

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

Pre-made components

Customized components

sources
setup files
.stylesheet

.c
.mdl
.h
.rpt

.rgs
.xml

.html
.doc
.pdf

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
RTW Build 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>');
    fprintf(fid, '<h4>Generated Source and Header Files:</h4>');
    fprintf(fid, '%s <br>', allSrcFileList(i));
    fprintf(fid, '</html>');
    fclose(fid);
end
```

Build Info API Demo

Generated Source and Header Files:
- rtwdemo_counter.h
- rtwdemo_counter_private.h
- rtwdemo_counter_types.h
- rtwtypes.h
- ert_main.c
- rtwdemo_counter.c

BuildInfoReport.html

```matlab
>> 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

```
>> 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