The MathWorks

MATLAB® & SIMULINK®

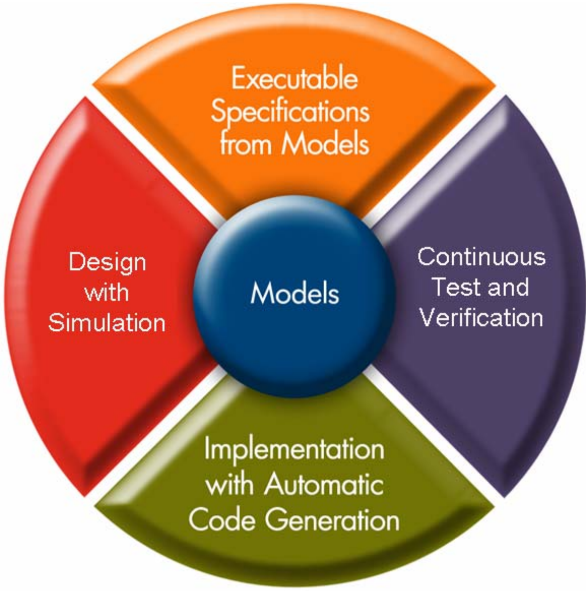
Production Code Generation Introduction and New Technologies

Tom Erkkinen

Embedded Applications Manager

The MathWorks, Inc.

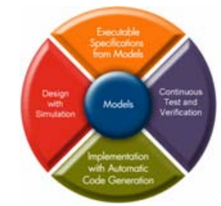
Agenda

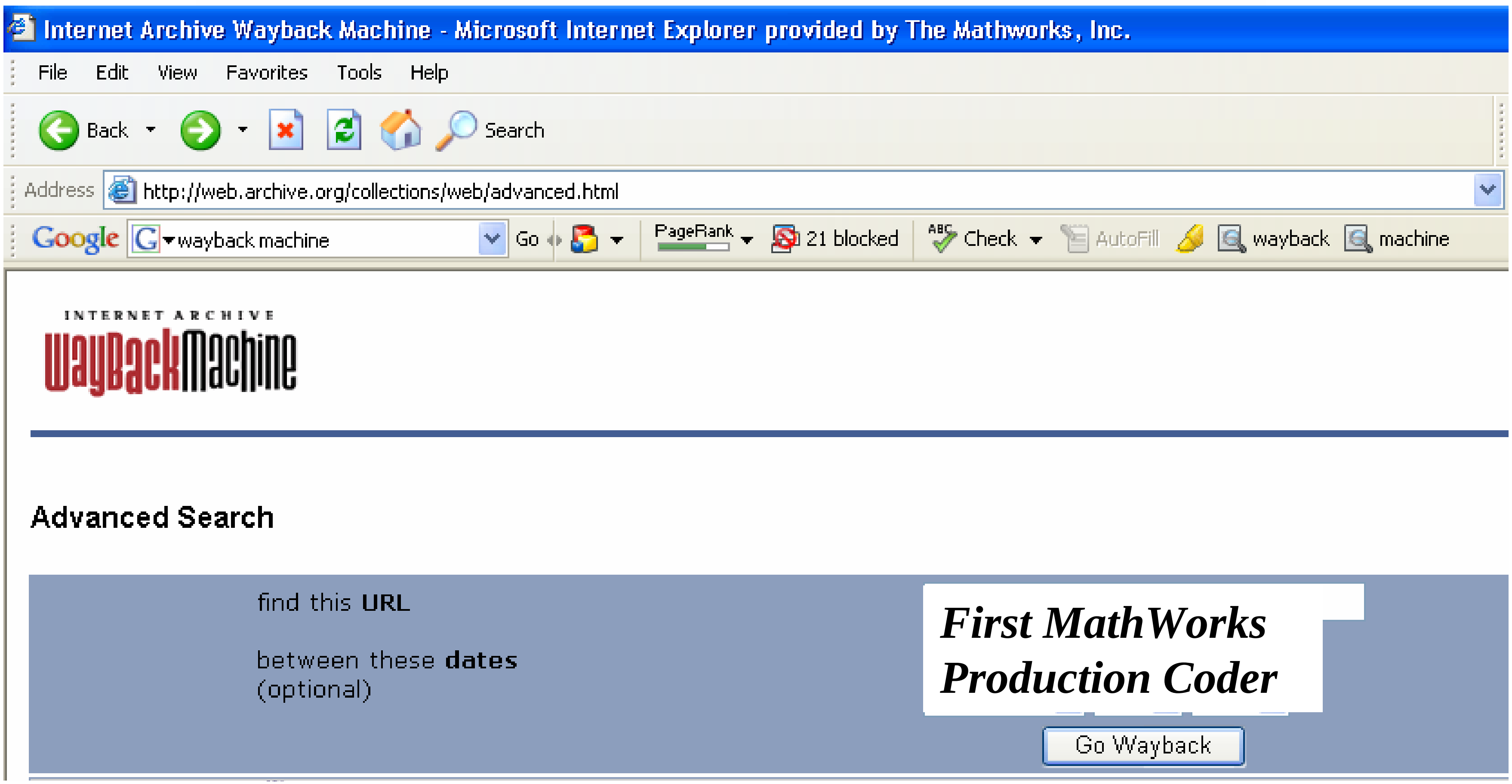


Historical Review

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

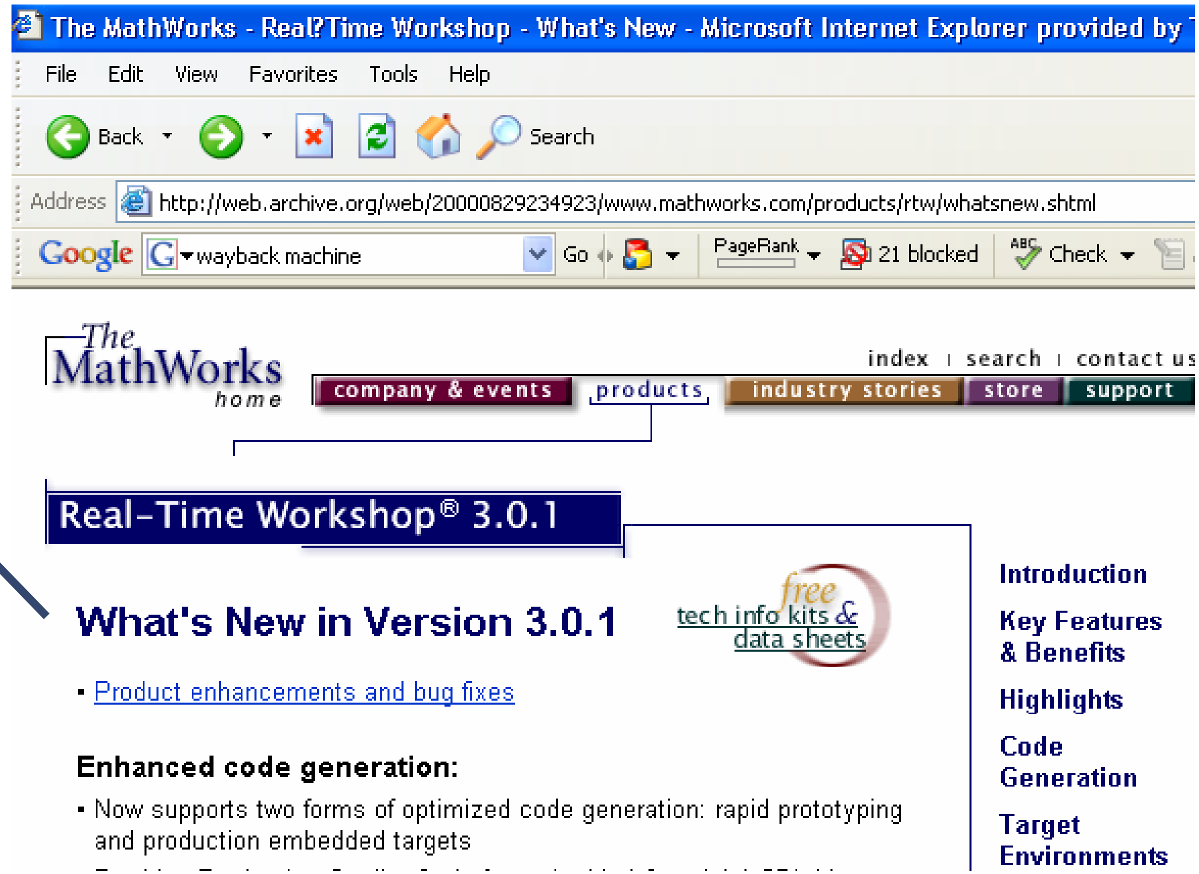
Wayback Machine (www.archive.org)



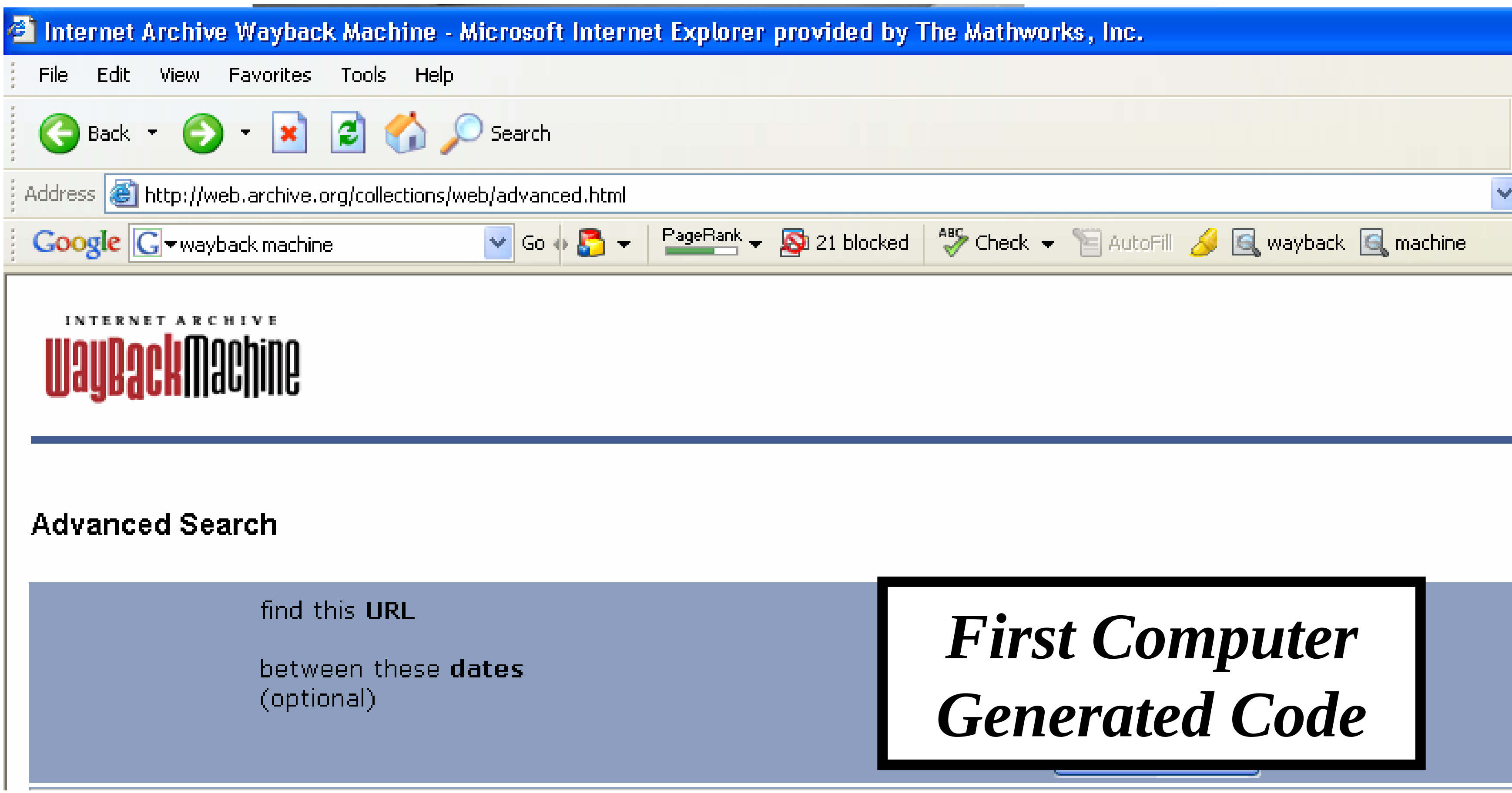


Simulink Code Generation - 1999

"Now supports two forms of code generation:



NACA “Computers” - 1949



First Computer Generated Code

Code Generation – 1990s

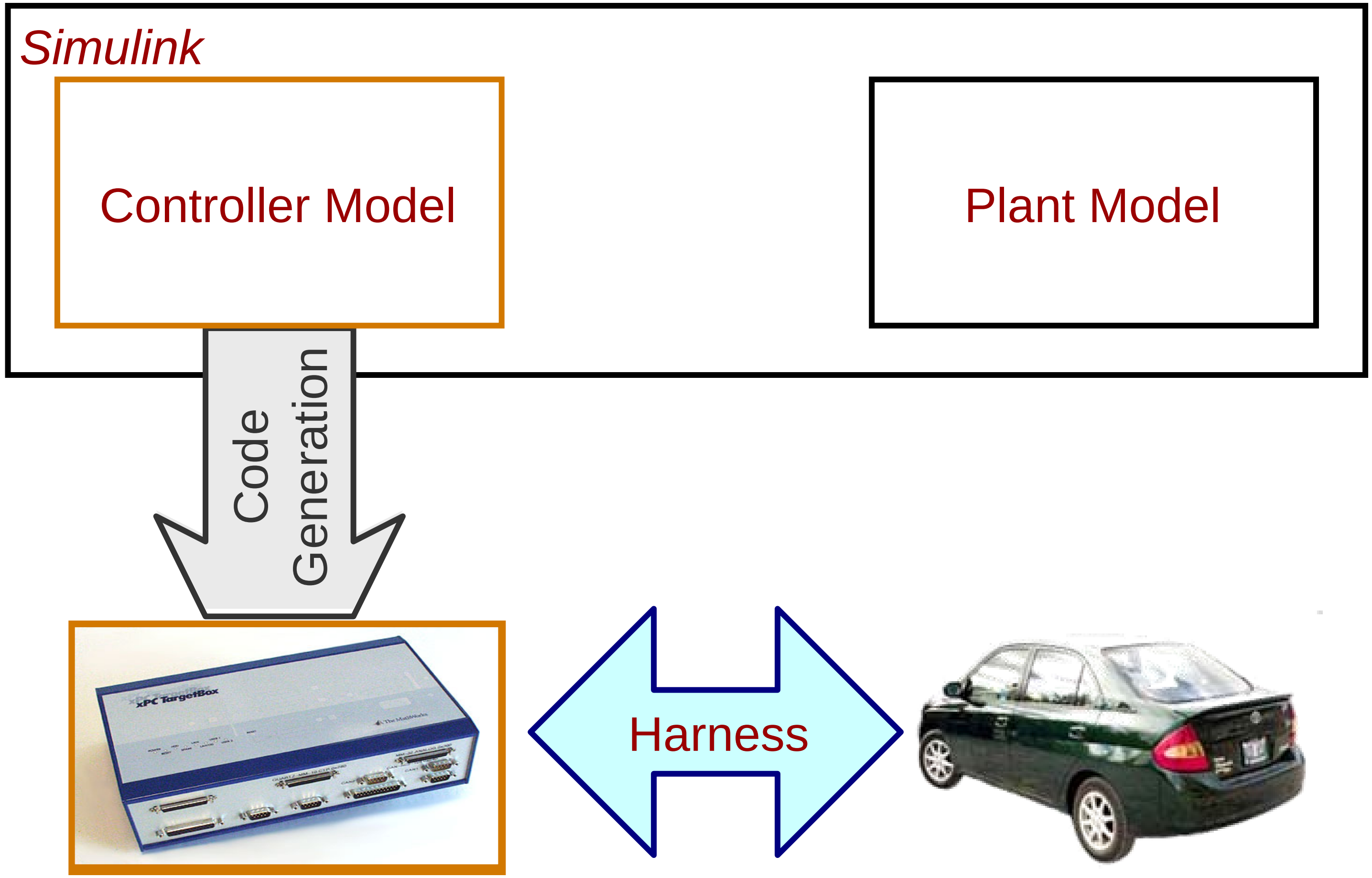
Simulink and Stateflow
Algorithm and System Design

Real-Time Workshop® and Stateflow® Coder

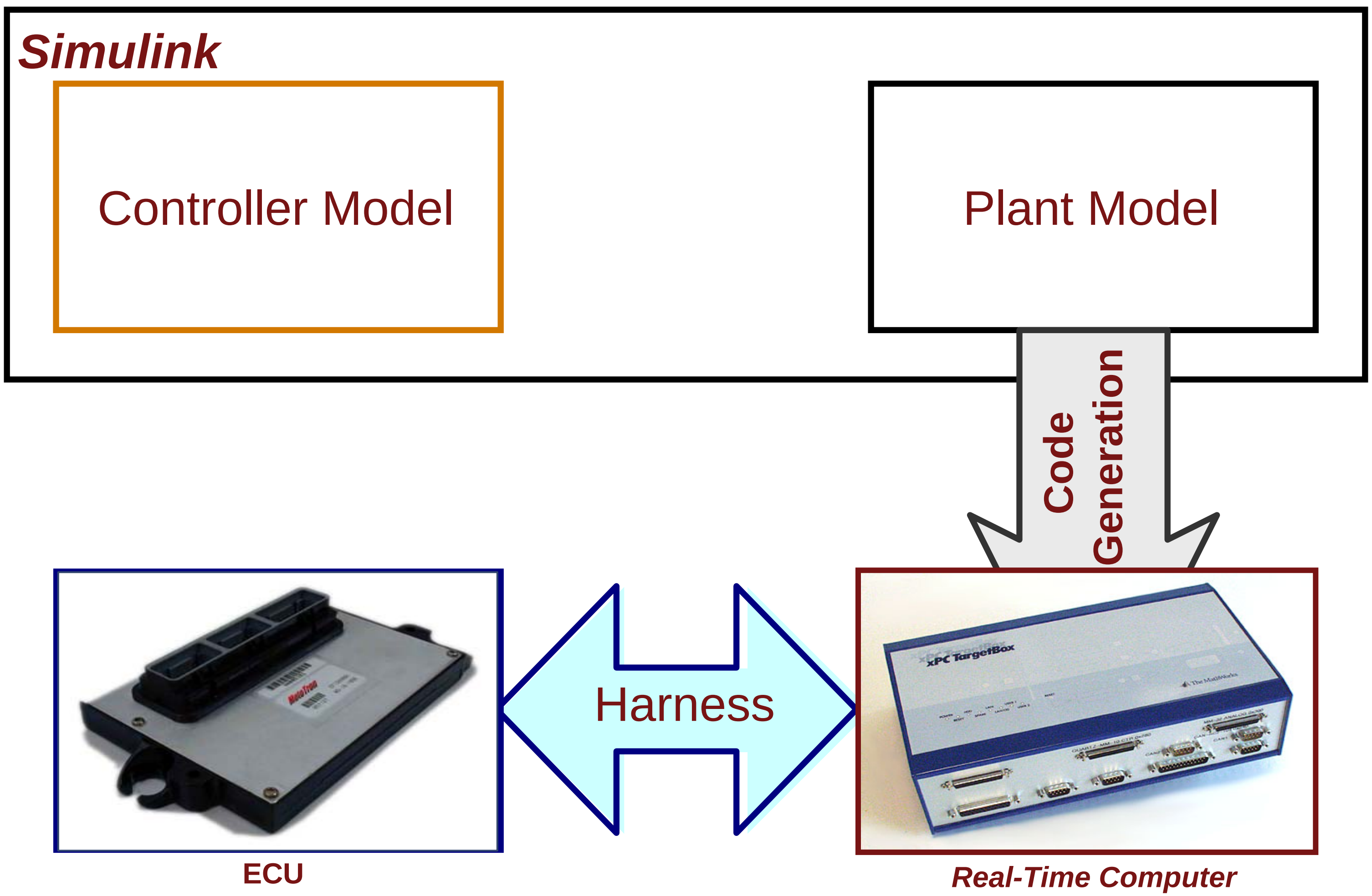
- Generate code from Simulink and Stateflow that is easy to interact, tune, and experiment with
  - Simulation Acceleration
  - Rapid Prototyping
  - Hardware in Loop (HIL)
  - Embedded deployment

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

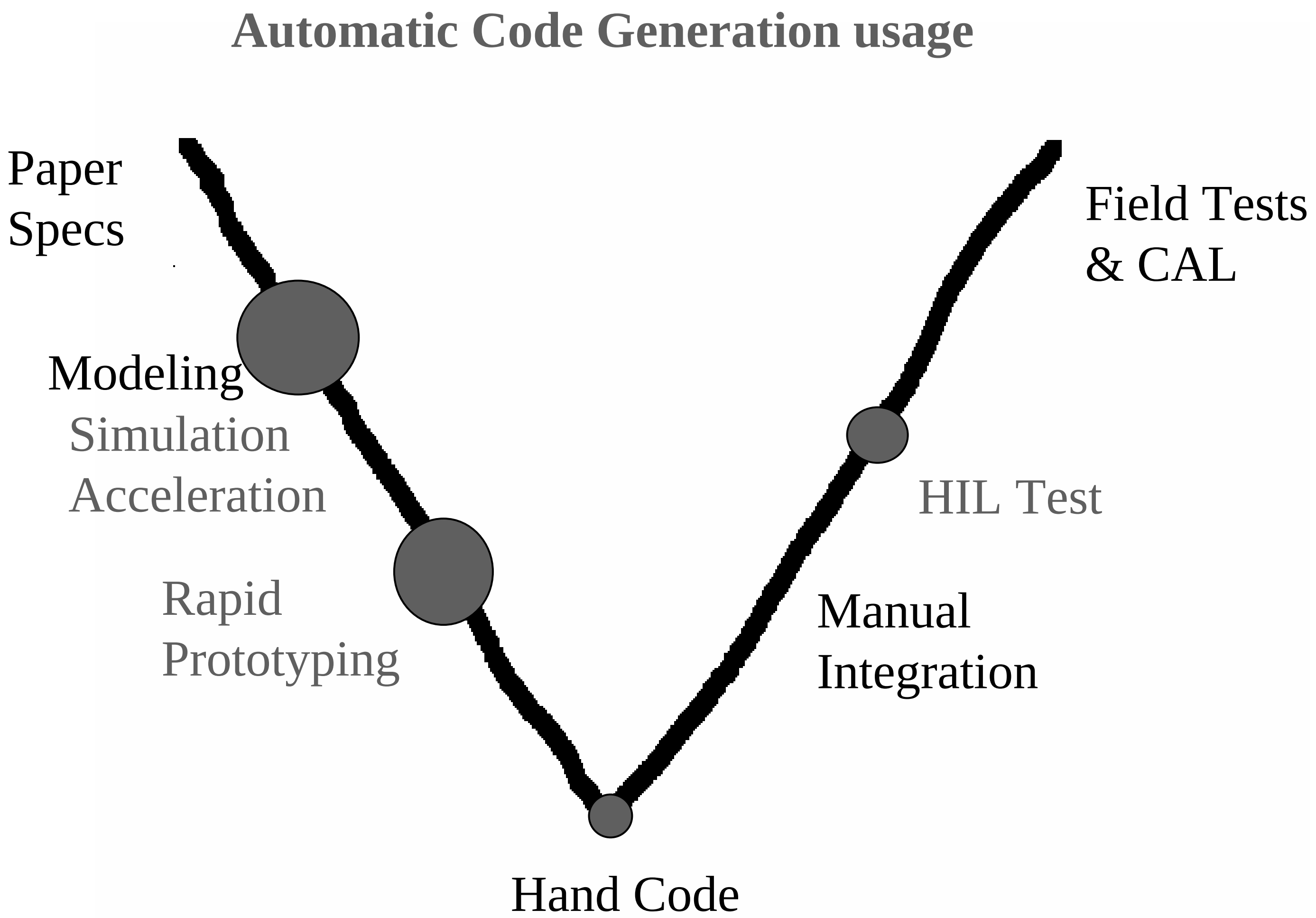
Rapid Prototyping



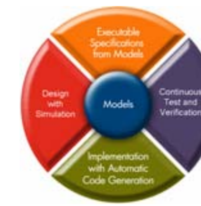
Hardware in Loop (HIL)

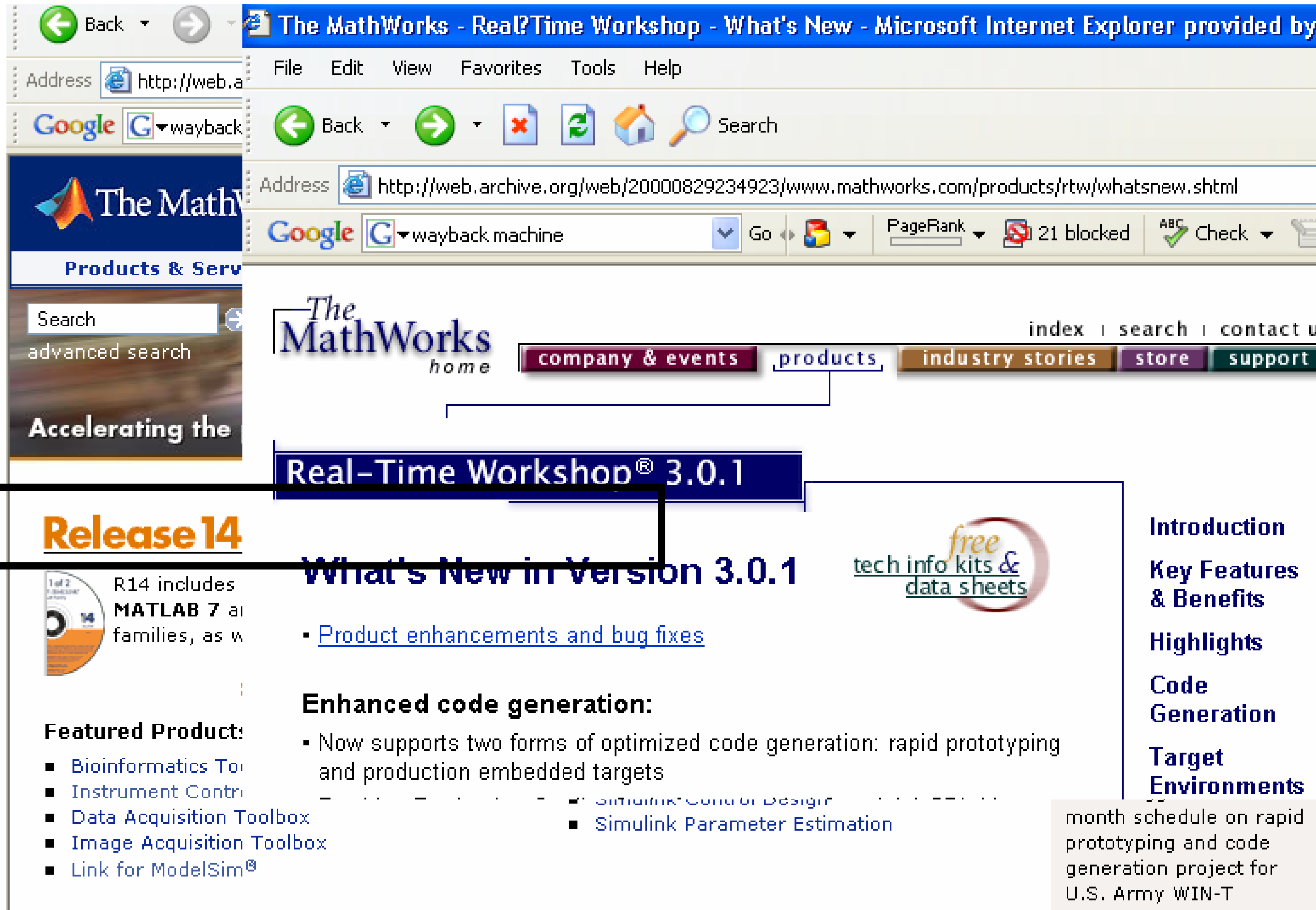


ECU Development – 1990s

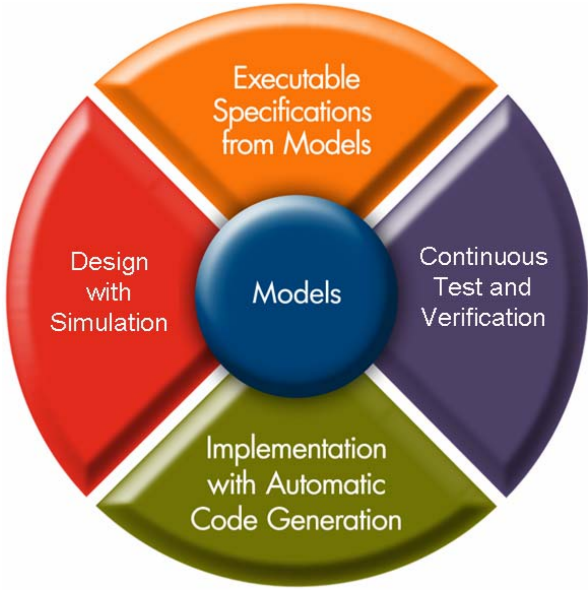


Simulink Code Generation - 1999





Agenda



Historical Review

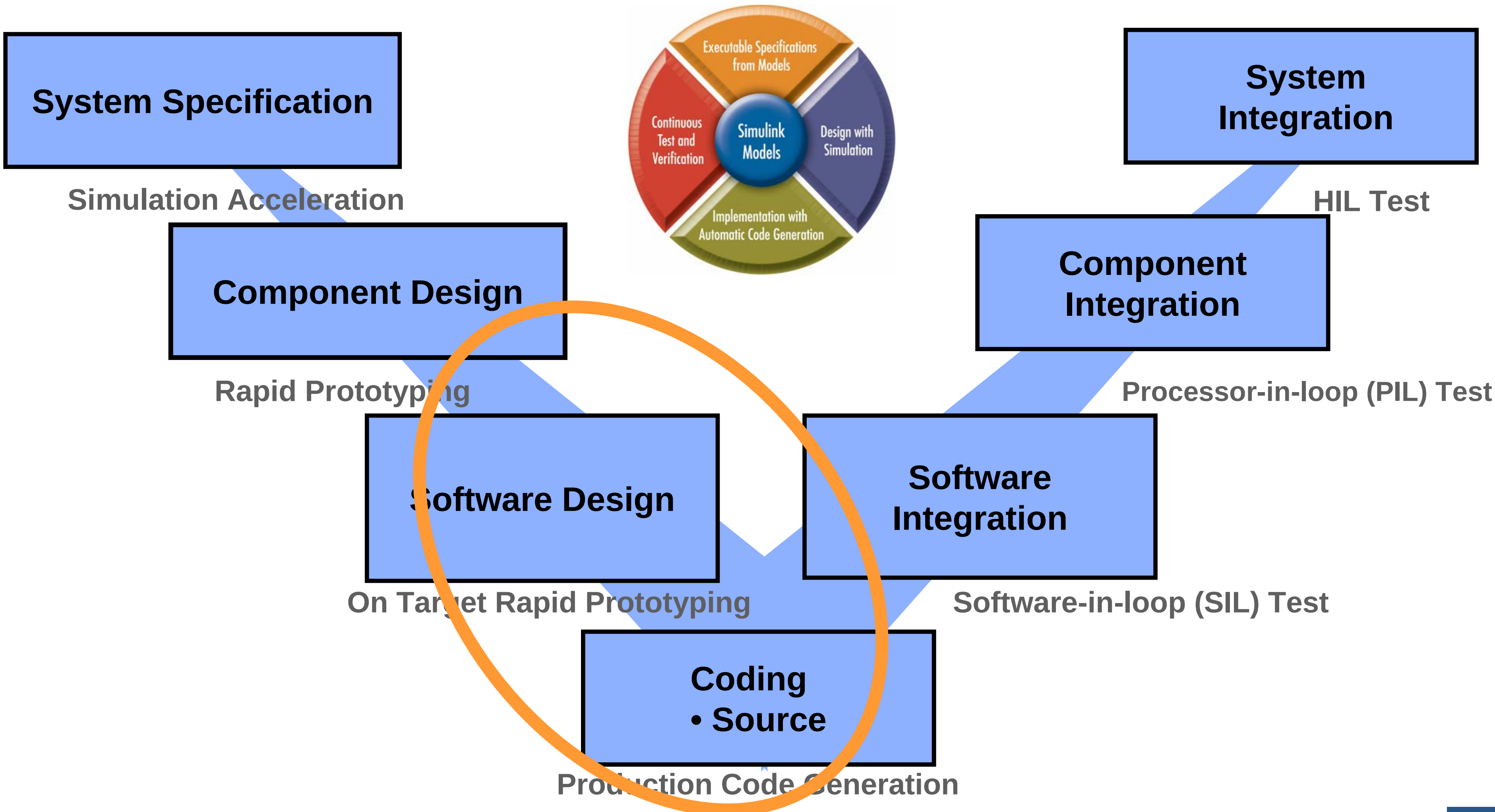
- Code Generation – 1999 (Release 11)
- Code Generation – 2005 (Release 14)
- Code Generation and Verification – R2007a

New features

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

ECU Development – 2005

Using Model-Based Design with Automatic Code Generation



Code Generation – 2005

Simulink and Stateflow
Algorithm and System Design

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.

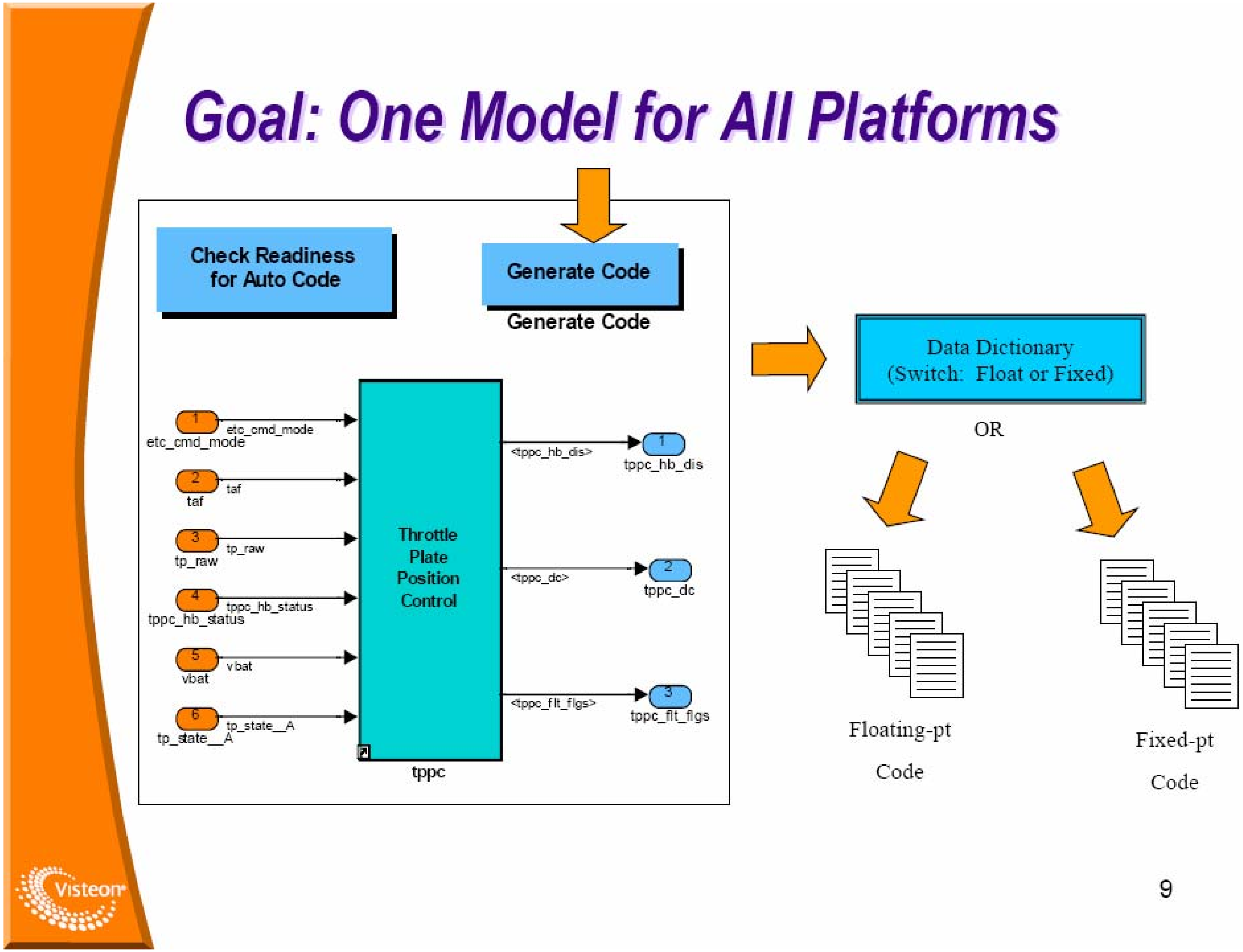
Keys to Success:

- Proficiency of users – requires mentoring by experts
- Standardized model style (especially tuned for code generation)
- Visteon custom tools

Visteon

4

MAC 2005


Goal: One Model for All Platforms
Check Readiness for Auto Code
Generate Code
Generate Code
Data Dictionary
(Switch: Float or Fixed)
OR
1
etc_cmd_mode
etc_cmd_mode
2
taf
taf
3
tp_raw
tp_raw
4
tppc_hb_status
tppc_hb_status
5
vbat
vbat
6
tp_state__A
tp_state__A
Throttle Plate Position Control
tppc
<tppc_hb_dis>
1
tppc_hb_dis
<tppc_dc>
2
tppc_dc
<tppc_flt_flgs>
3
tppc_flt_flgs
Floating-pt Code
Fixed-pt Code
Visteon
9

MAC 2005

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**

Visteon

Benefits of Fixed-point Code Generation

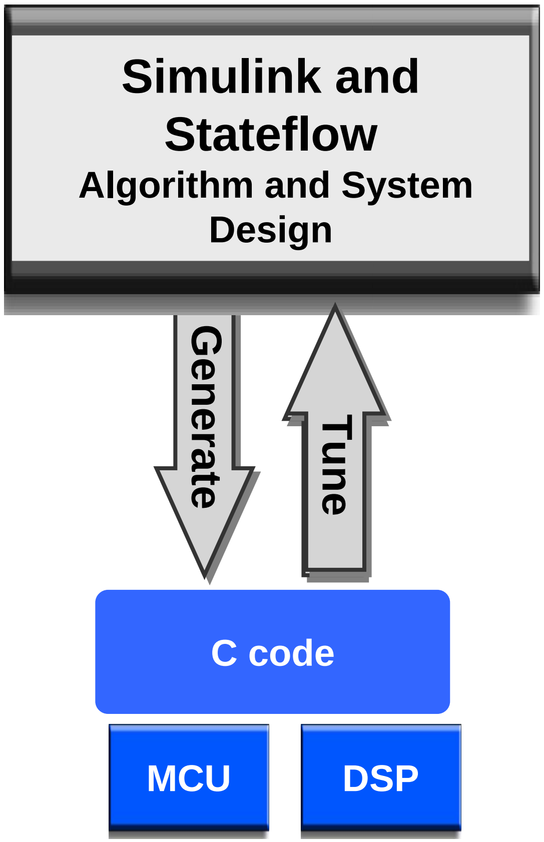
- **Significant savings in development time**
  - Pilot Project - real work took about 2-3 months (excluding time spent on resolving tools issues) Hand code would have taken at least 6 months
- **Fixed-point computation functions/macros**
  - Bulk of them already built by Mathworks
- **Easier unit testing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

Visteon

Code Generation – 2005

Real-Time Workshop® Embedded Coder

- Generates efficient code that can be customized to look like hand code for
  - Production Code Generation

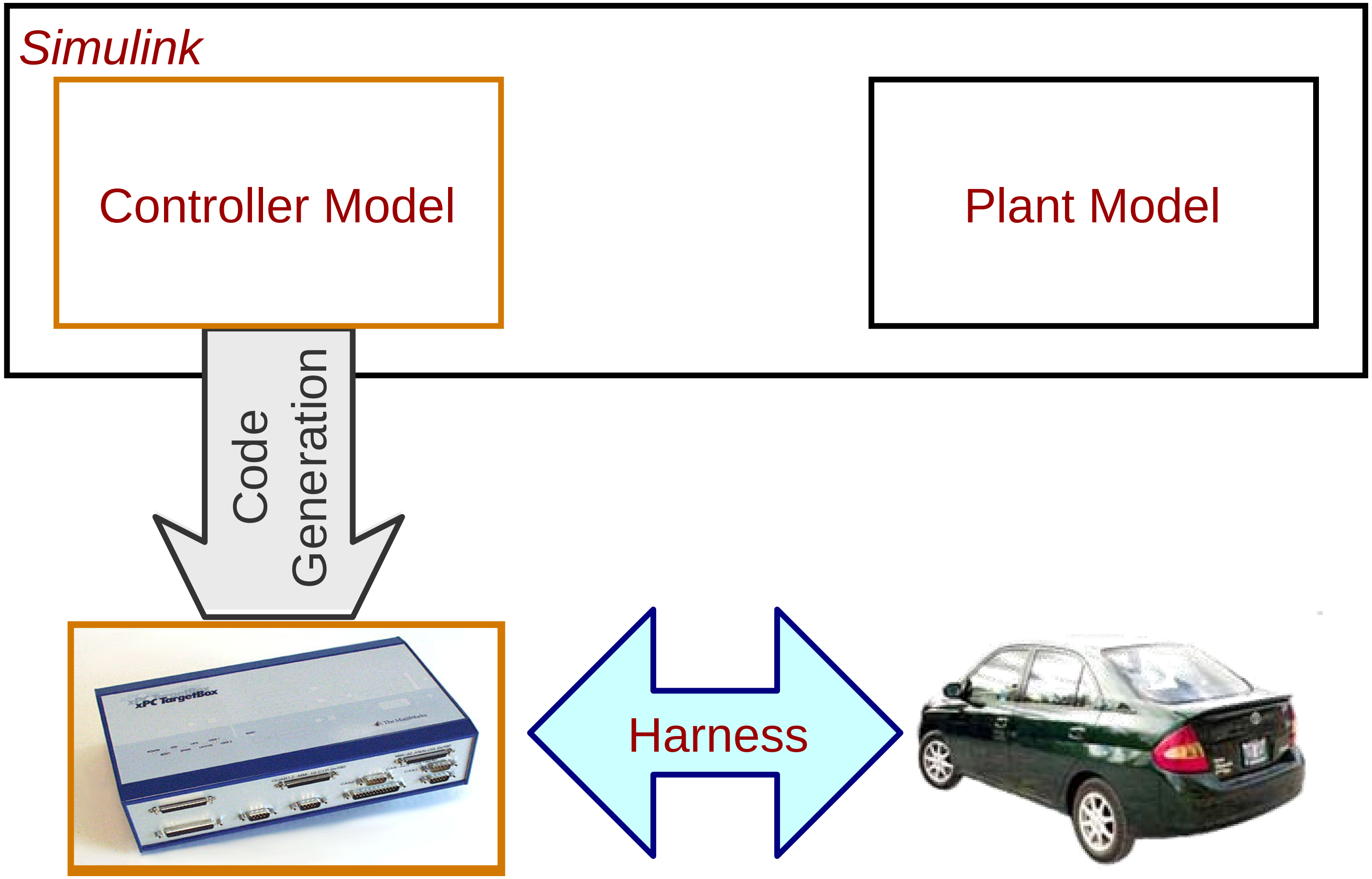


Embedded Targets

- 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



Rapid Prototyping Comparison

	Traditional Bypass	On-Target
Hardware (Cost)	From off-the-shelf PCs To custom bypass HW	From Existing ECUs To eval board HW
Model Flexibility	More emphasis	Less emphasis
Code Efficiency	Less emphasis	More emphasis
Purpose	New ideas, green field research	Refine designs, production focused

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

	Hand Code	Auto-code	Percent Change
Calibration ROM	9464	9464	0%
Code ROM	2952	2900	-1.76%
RAM	240	238	-0.83%

More Efficient Than Hand

MAC 2005

2005 Automatic Code Generation Objectives

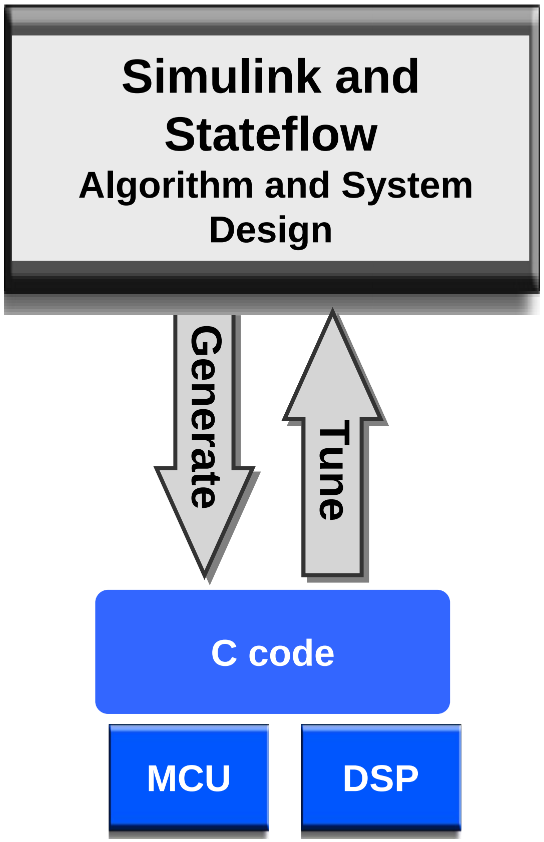
- Re-evaluate hand code process and auto-code process to reduce overhead
- Address automatic code generation for multiple large functions in a single model file
- Develop a seamless process for testing the model and the generated code
- Develop on-target rapid prototyping process using Mathworks Embedded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

GM MathworksMIAC21June05.ppt 15

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)

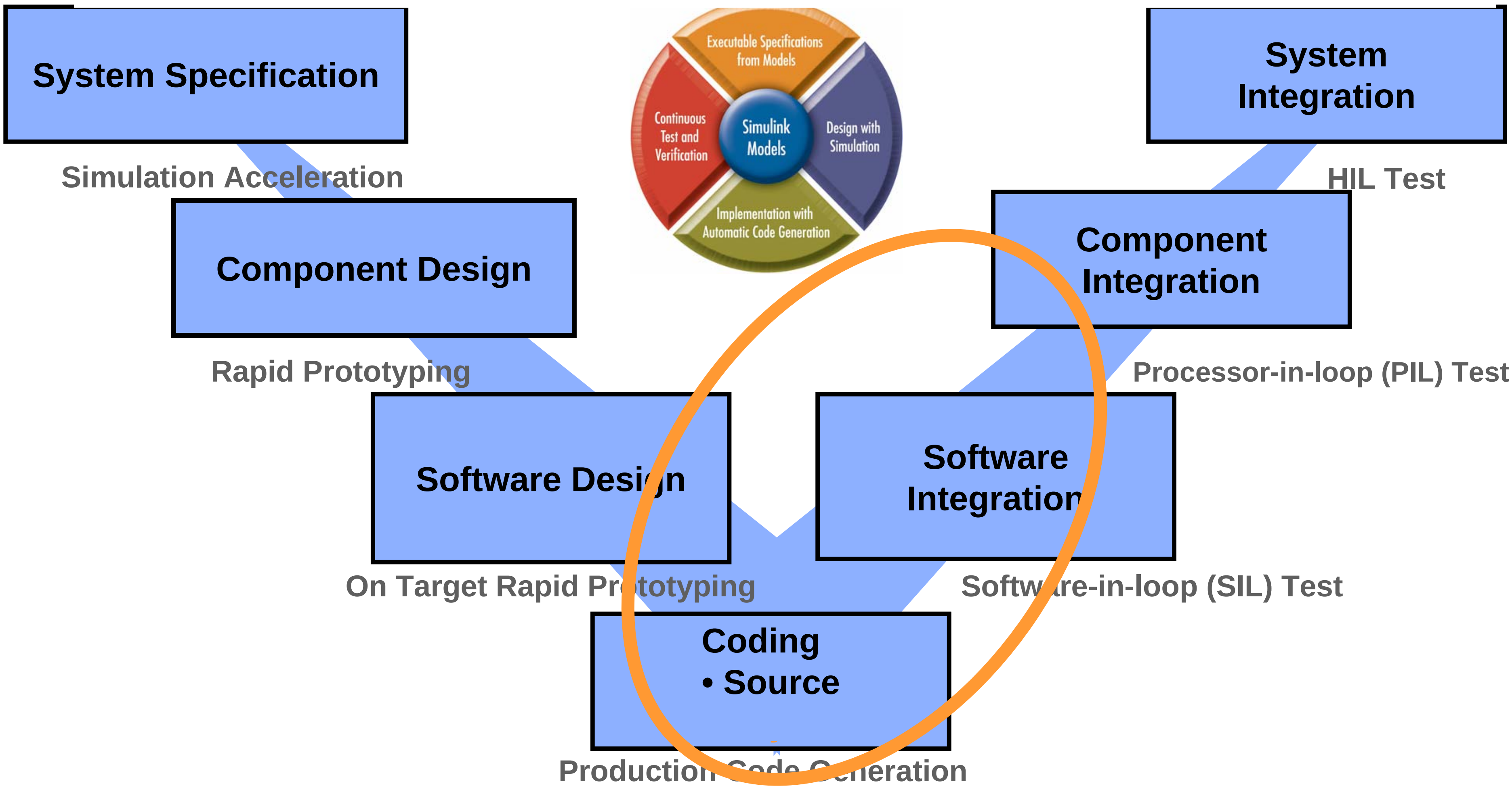


Embedded Targets

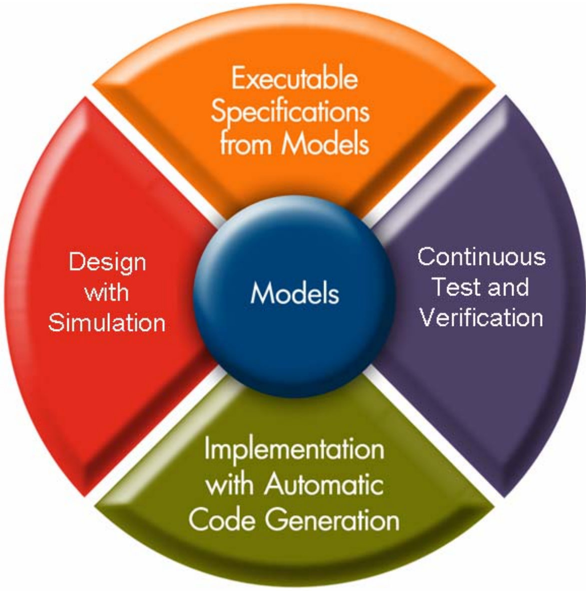
- Provide target specific blocks/features for
  - On-Target Rapid Prototyping
- Some PIL support (MPC5xx focused)

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)



Agenda



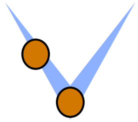
Historical Review

- 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



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

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

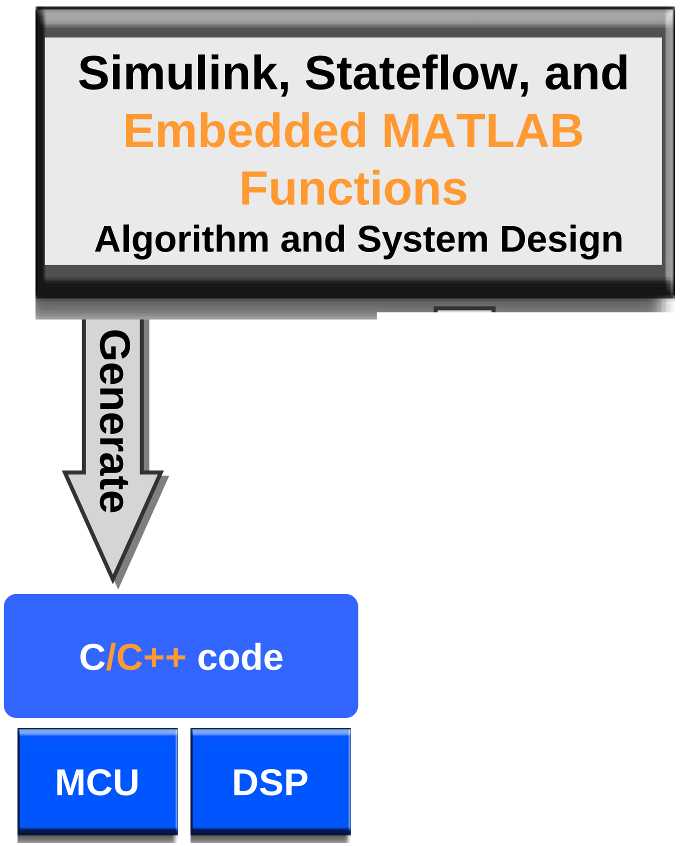
Code Generation – R2007a

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

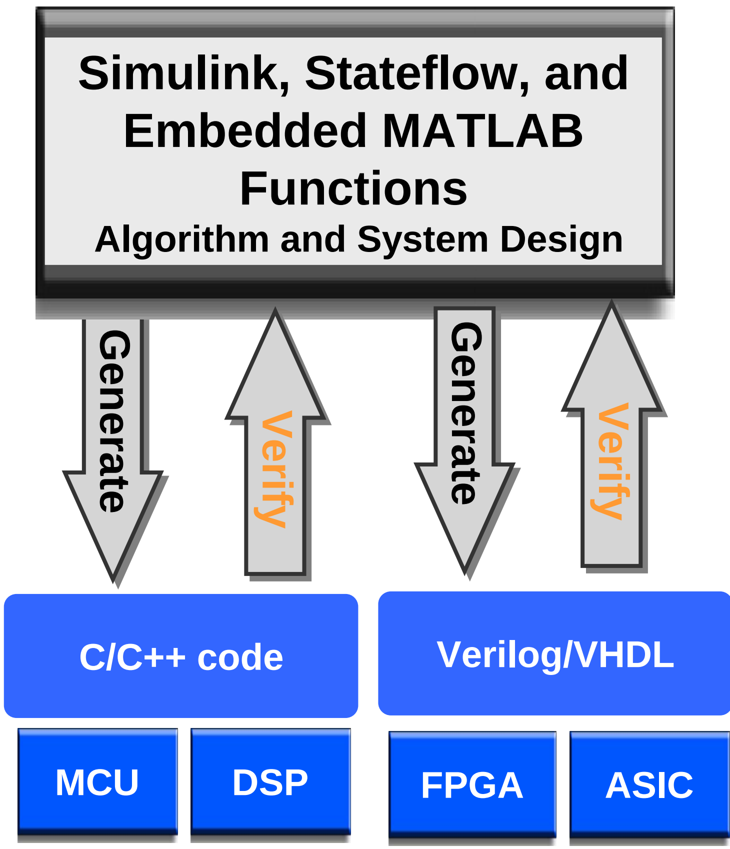
- C and C++ Production Code Generation
- SIL Options

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

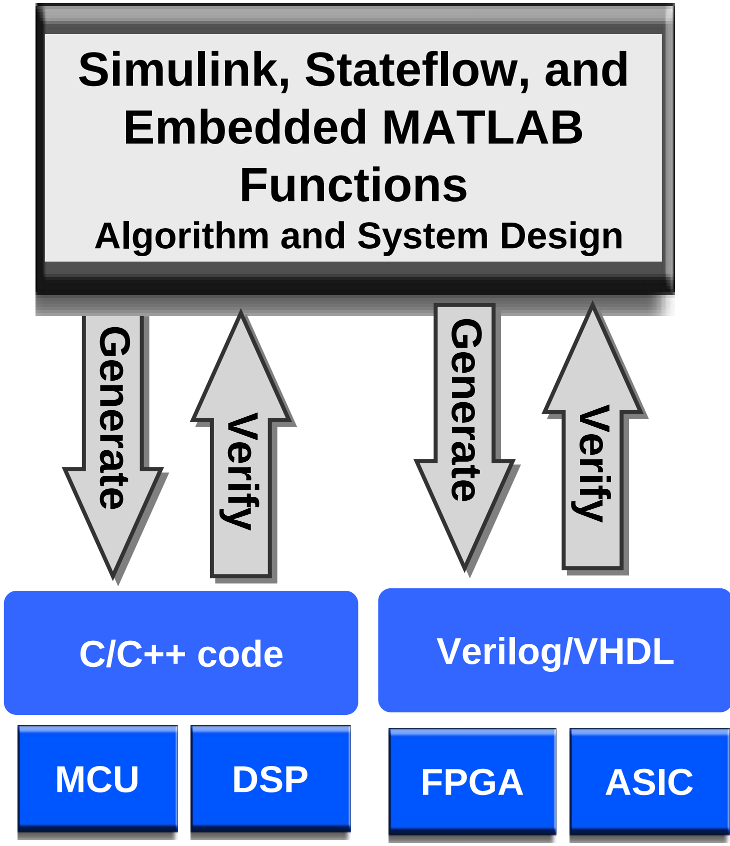
- C and C++ Production Code Generation
- SIL Options
  - Emulated or Actual

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

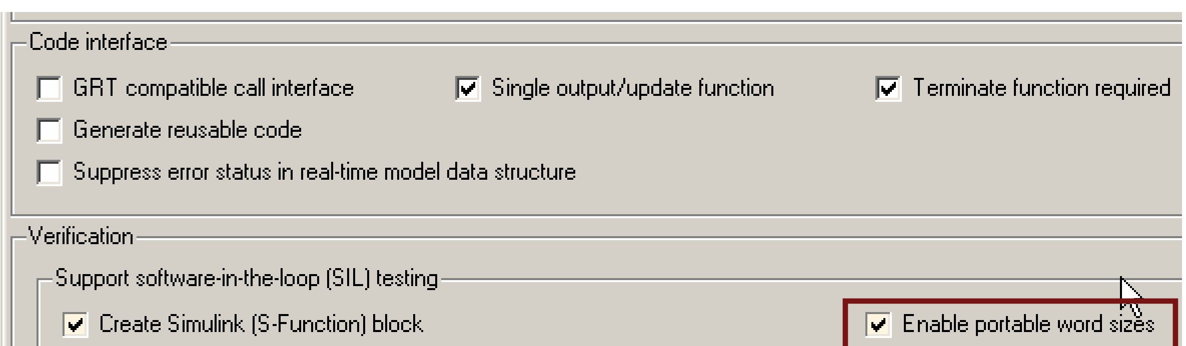
- Word sizes for the host and target platforms often differ
- Fixed sizes inhibit software-in-the-loop (SIL) testing of production intent code

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



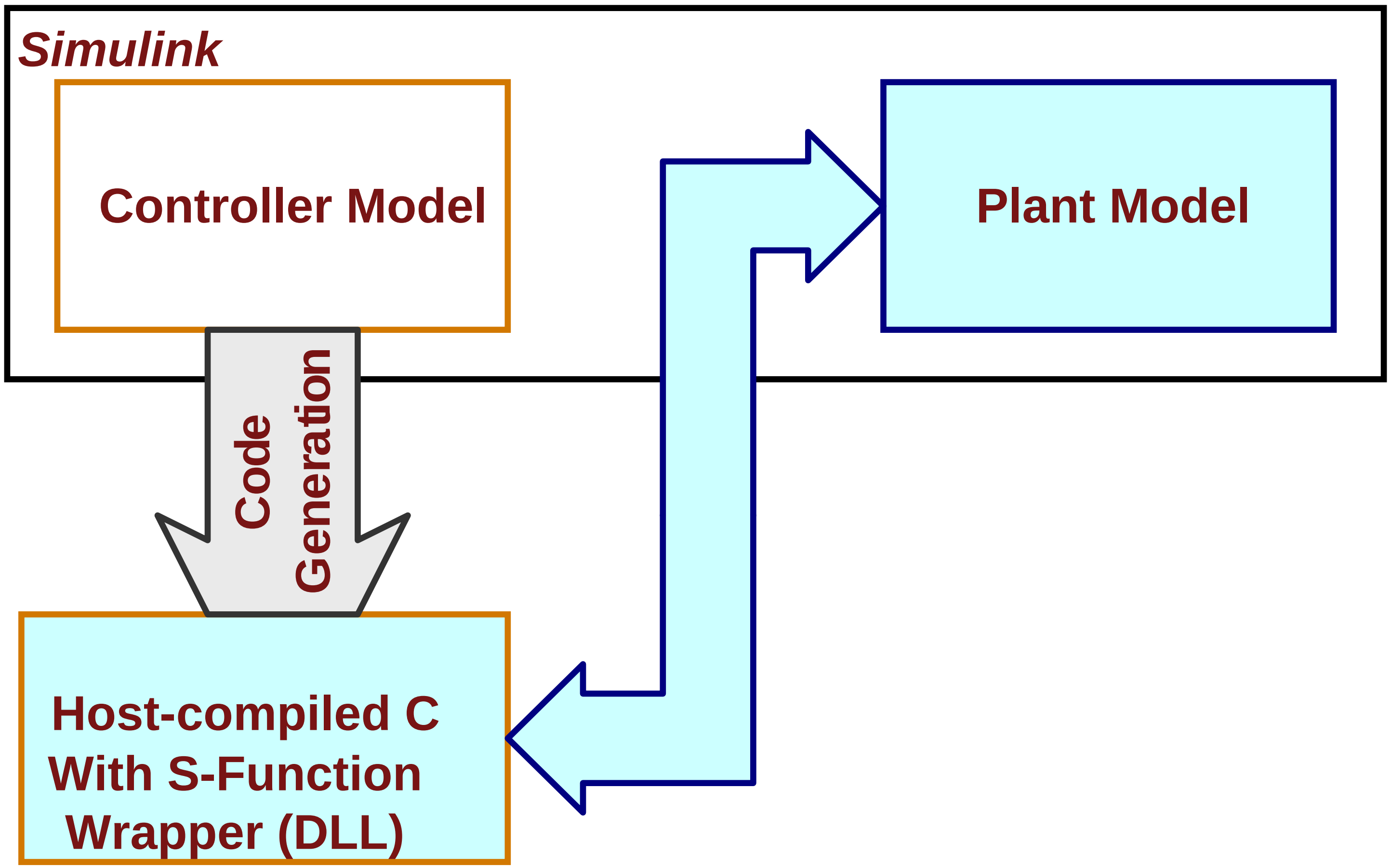
SIL

```
#ifdef 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



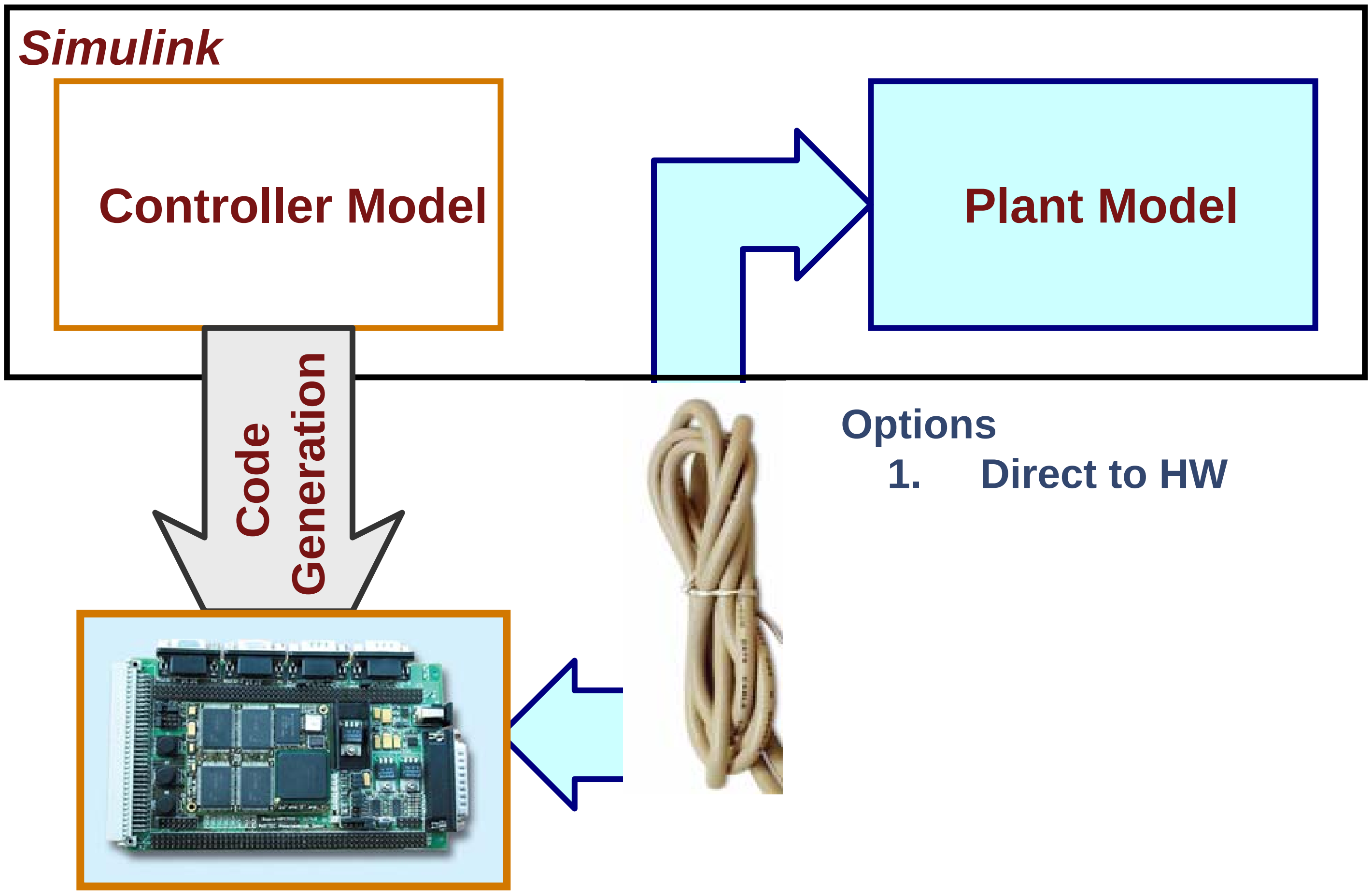
Real-Time Workshop Embedded Coder SIL option

SIL vs. HIL Comparison

Description	SIL	HIL
Purpose	Verify Source Code Component	Verify Complete System Functionality
Platform (Plant, ECU)	Host, Host	Target, Target
Target Fidelity	Emulated: same source code, not bit accurate Actual: Different source code, bit accurate (fix-pt) (LOW)	Same executable code; bit accurate (fix-pt); cycle accurate; emulated I/O (HIGH)
Convenience	Desktop convenient; executes just in Simulink; no HW cost (HIGH)	Executes in test bench or lab; $$ for processor, ECU, I/O, cables (LOW)

PIL – Direct HW

For MPC5xx Microprocessor Hardware



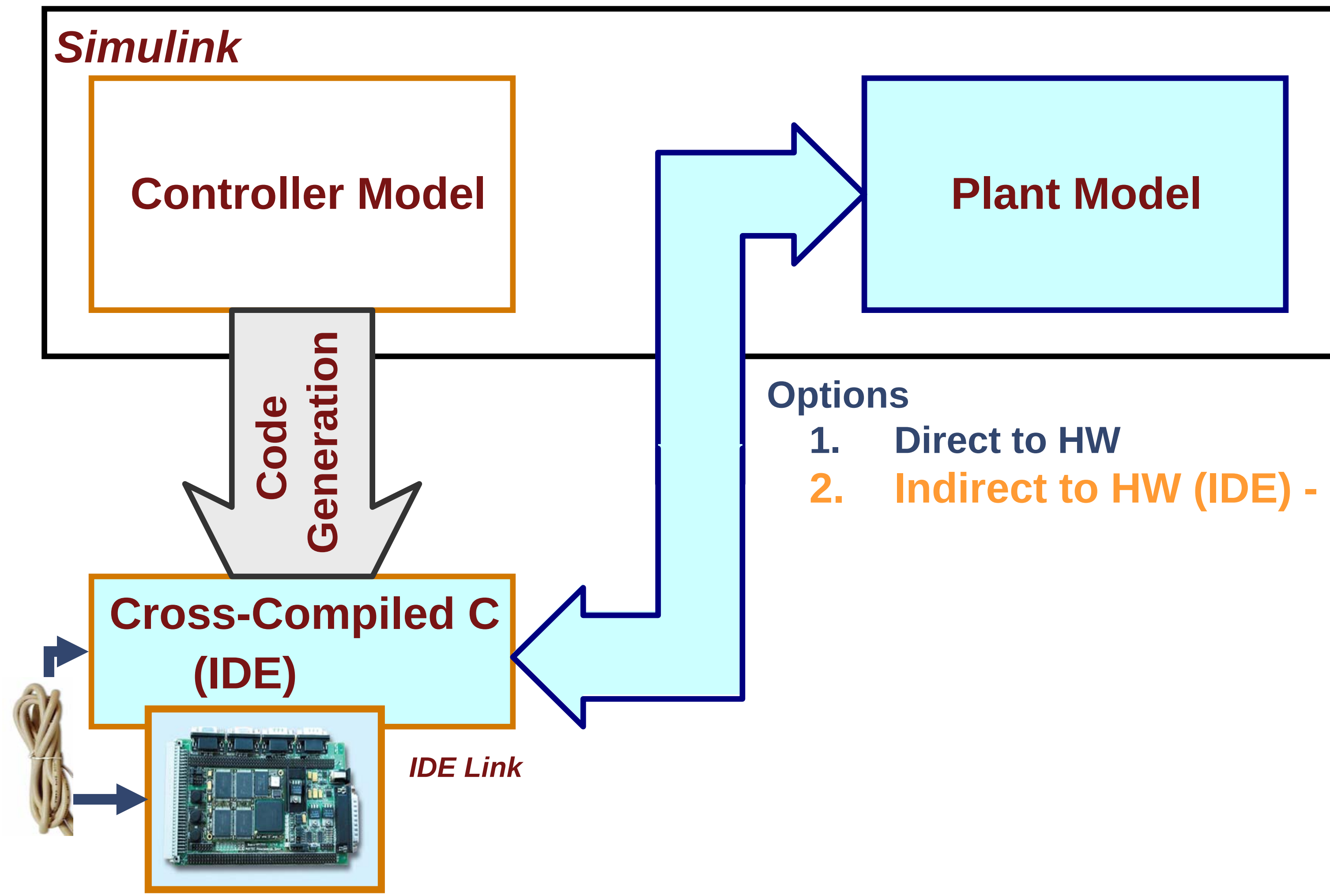
Options

1. Direct to HW

Target for Freescale™ MPC5xx

PIL – Indirect HW

For Many Microprocessors Supported by IDE

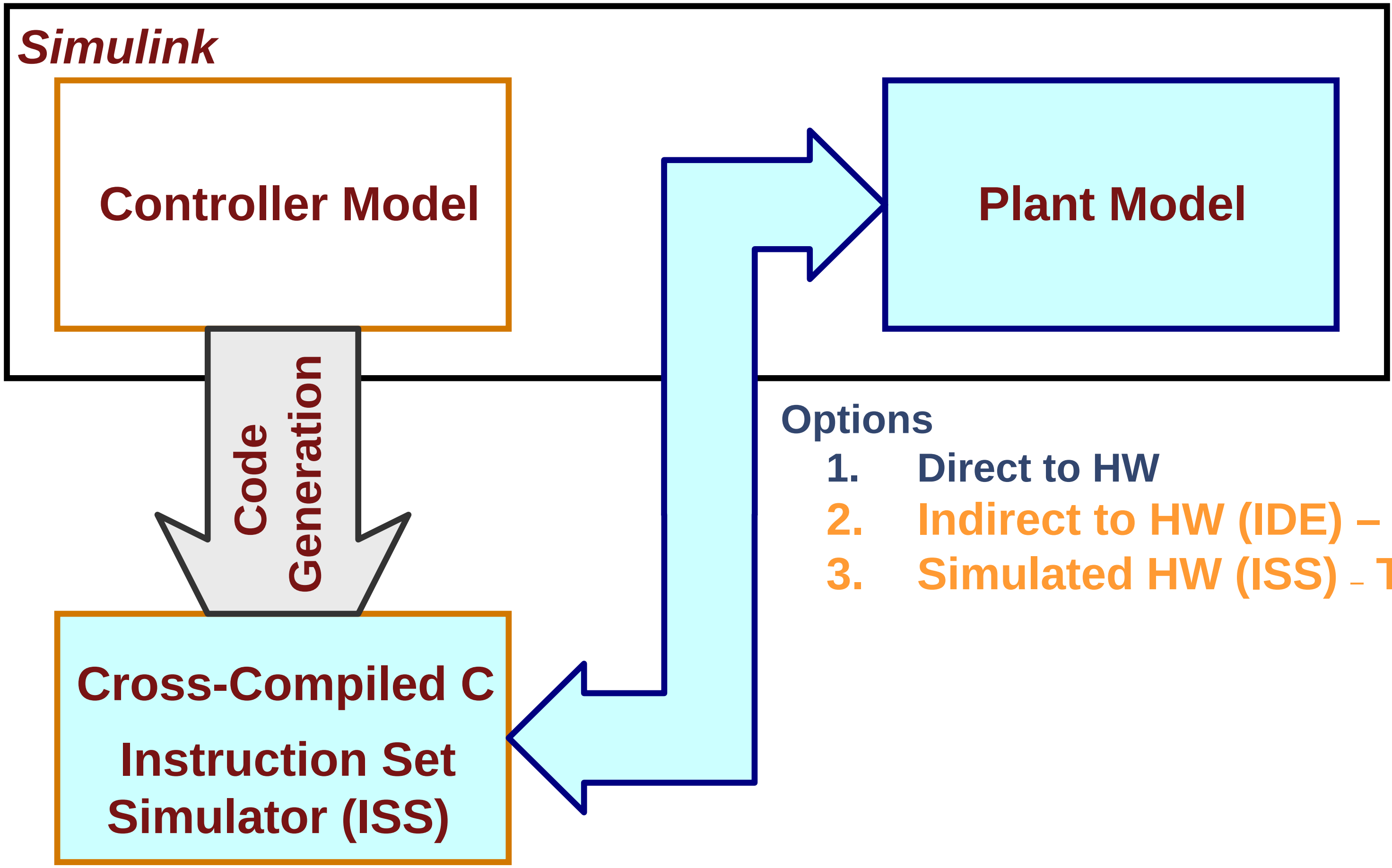


Options

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

IDE Link

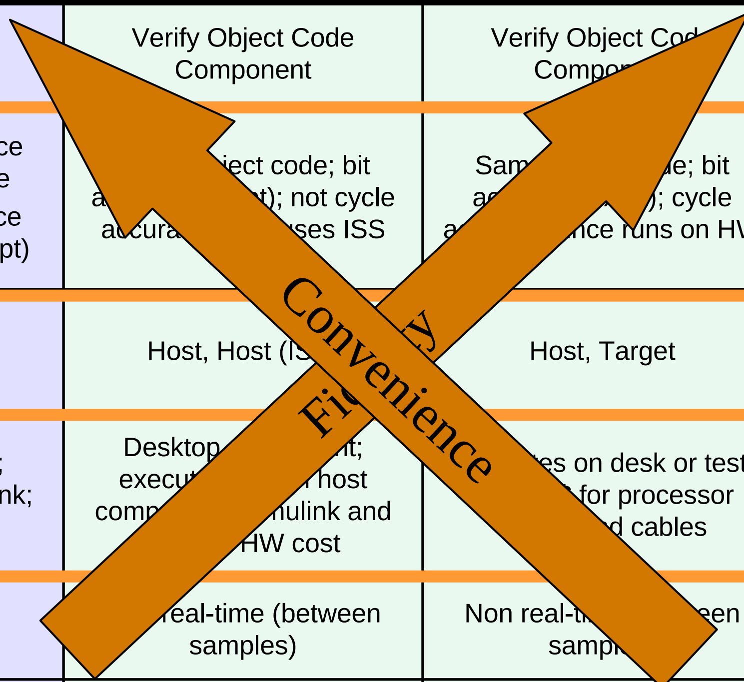
Processor-In-the-Loop Comparison

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

*ISS Instruction Set Simulator

In-the-Loop Comparison

Description	SIL	PIL (ISS)	PIL (HW)	HIL
	Software-in-the-Loop	**Processor-in-the-Loop**		**Hardware-in-the-Loop**
Purpose	Verify Source Code Component	Verify Object Code Component	Verify Object Cod… Compo…	Verify Complete System Functionality
Fidelity	Emulated: same source code, not bit accurate Actual: Different source code, bit accurate (fix-pt)	…ject code; bit a…t); not cycle accura… …ses ISS	Sam… …de; bit ac… …); cycle a… …nce runs on HW	Same executable code; bit accurate (fix-pt); cycle accurate; emulated I/O
Platform (Plant, ECU)	Host, Host	Host, Host (IS…	Host, Target	Real-Time System, Target
Convenience	Desktop convenient; executes just in Simulink; no HW cost	Desktop …nt; execu… …host comp… …mulink and …HW cost	…tes on desk or test … for processor …d cables	Executes in test bench or lab; $$ for processor, ECU, I/O, cables
Real-Time	Non real-time	…real-time (between samples)	Non real-ti… …een sampl…	Hard real-time
Engineers	Systems or Software Engineers	Software or Test Engineers	Software or Test Engineers	Systems or Test Engineers



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 metho
- 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!

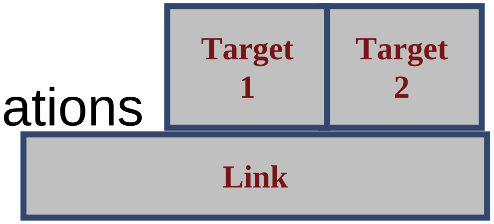
Link and Target Products

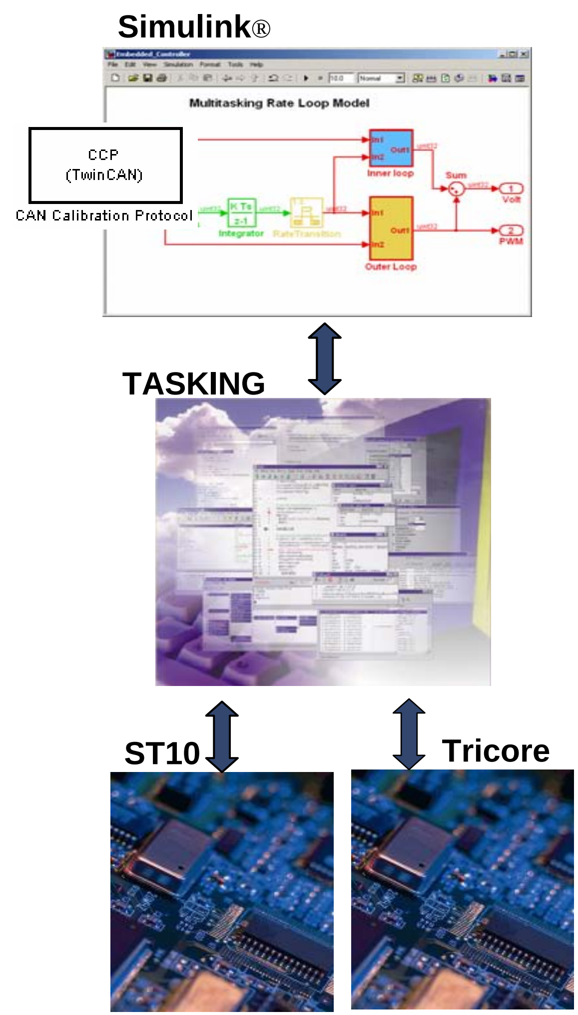
Links

- 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

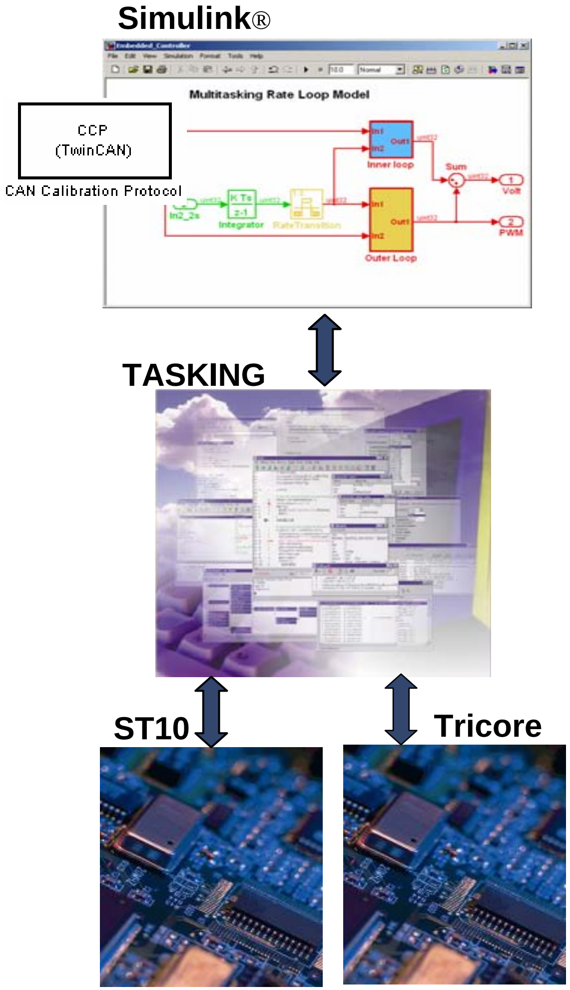




Links and Targets Product Providers

Availability

- MathWorks
  - Products
  - Consulting
- MATLAB Central
- Third Parties



MATLAB Central

MATLAB Central > File Exchange > Production Code

Production Code (18 Files)

Click on column heading to sort by column

Downloads (last 30 days)	Title	Submitted	Rating (5=best)
248	Embedded Coder Robot NXT Demo Offers an enjoyable Model-Based Design experience using Simulink models with Lego robots Author: Tom Erkkinen Category: Production Code	2006-12-15	4.25 4 reviews
77	TLC :Quick start guide Quick start guide to writing and understanding TLC. Author: Nitin Skandan Category: Production Code	2005-04-18	4.0
73	MPC555 Motor Control Function Blockset Additional I/O blocks for MPC555 target - specifically targetting TPU functionality Author: Edward Hartley Category: Production Code	2006-03-20	4.0 3 reviews
69	Address Specific Parameter Custom Storage Class MemMap: A custom storage class that allows the user to specify the memory addresses of parameters in Author: Pete Maloney Category: Production Code	2007-05-14	0 reviews
52	Using MathWorks tools to apply Model-Based Design for DO-178B Applications An example workflow for Model-Based Design to generate code for safety critical applications. Author: Vinod Cherian Category: Production Code	2007-03-20	5.0 1 review
48	Integrating Processor-Specific Code with Model-Based Design of Embedded Systems Paper and models describing a technique for integrating processor-specific code Author: Tom Erkkinen Category: Production Code	2007-02-12	0 reviews

36	External I/O and State Information Block This block generates an extra file during code generation containing external Inputs, outputs an Author: Roger Aarenstrup Category: Production Code	2006-07-19	5.0 1 review
35	Processor In the Loop with Link for TASKING This demo shows how Processor In the Loop (PIL) testing can be used for verification of automaticall Author: David Maclay Category: Production Code	2007-03-09	0 reviews
32	Optimized Infineon TriCore Simulink Blocks for use with Link for TASKING Simulink blocks (FIR, FFT) optimized for the Infineon Tricore using the Infineon TriLib DSP library. Author: Jon Fielder Category: Production Code	2007-02-23	0 reviews
32	Model Assistant Tool Using the Model Assistant Tool, you can quickly configure a model for code generation. Author: Pete Szpak Category: Production Code	2002-12-06	2.0 1 review
31	Code Coverage Tool Code Coverage Tool measures C statement coverage in generated code. Author: Mark Walker Category: Production Code	2007-03-15	3.29 7 reviews
30	C-CAN drivers for use with Target for Infineon C166 Support for C-CAN hardware used on newer variants of the ST10 microcontroller Author: David Maclay Category: Production Code	2007-03-09	0 reviews

www.mathworks.com/matlabcentral/fileexchange

Third Party Links and Targets

Processors	C/C++ Support	Built-In Data Type Support	Link Products	Target Products	Third-Pa... Pr...
All processors	✔				
32-bit PowerPC	✔	✔			
8051 Compatible	✔	✔	Link for TASKING		
AMD K5/K6/Athlon	✔	✔			
ADI Blackfin	✔	✔	Link for A... Devices VisualDSP...		
ADI SHARC	✔	✔	Link for A... Devices VisualDSP++®		
ARM 7/8/9	✔	✔	Link for TASKING		GA...
Freescale MPC86xx, MPC7xxx, and MPC52xx (32-bit PowerPC)	✔	✔			
Freescale MPC55xx	✔	✔			Freescale Mototron
Freescale MPC5xx	✔	✔		Target for Freescale MPC5xx	Various

TI OMAP	✔	Link for Code Composer Studio
TI TMS470	✔	Link for Code Composer Studio

Select the manufacturer to see a full list of supported hardware

...quest Additional Hardware Interface Support

...Indicates Required Information

...Hardware Manufacturer

...omments

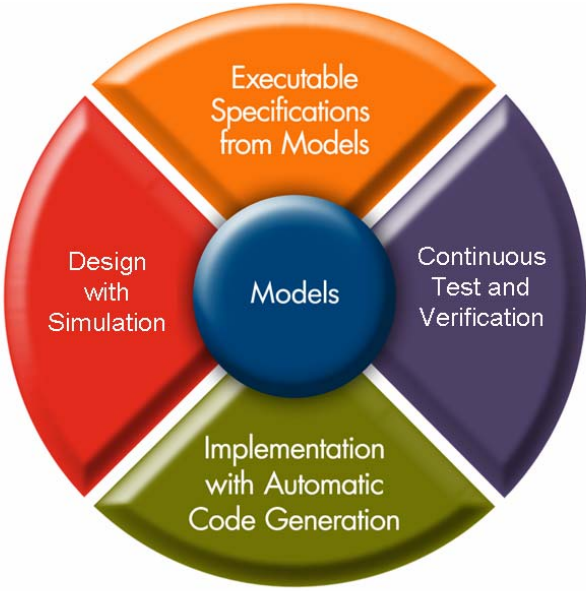
*E Mail (Format must be eg. name@yourdomain.com)

Submit

We will not sell or rent your personal contact information. See our privacy policy for details.

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

Agenda



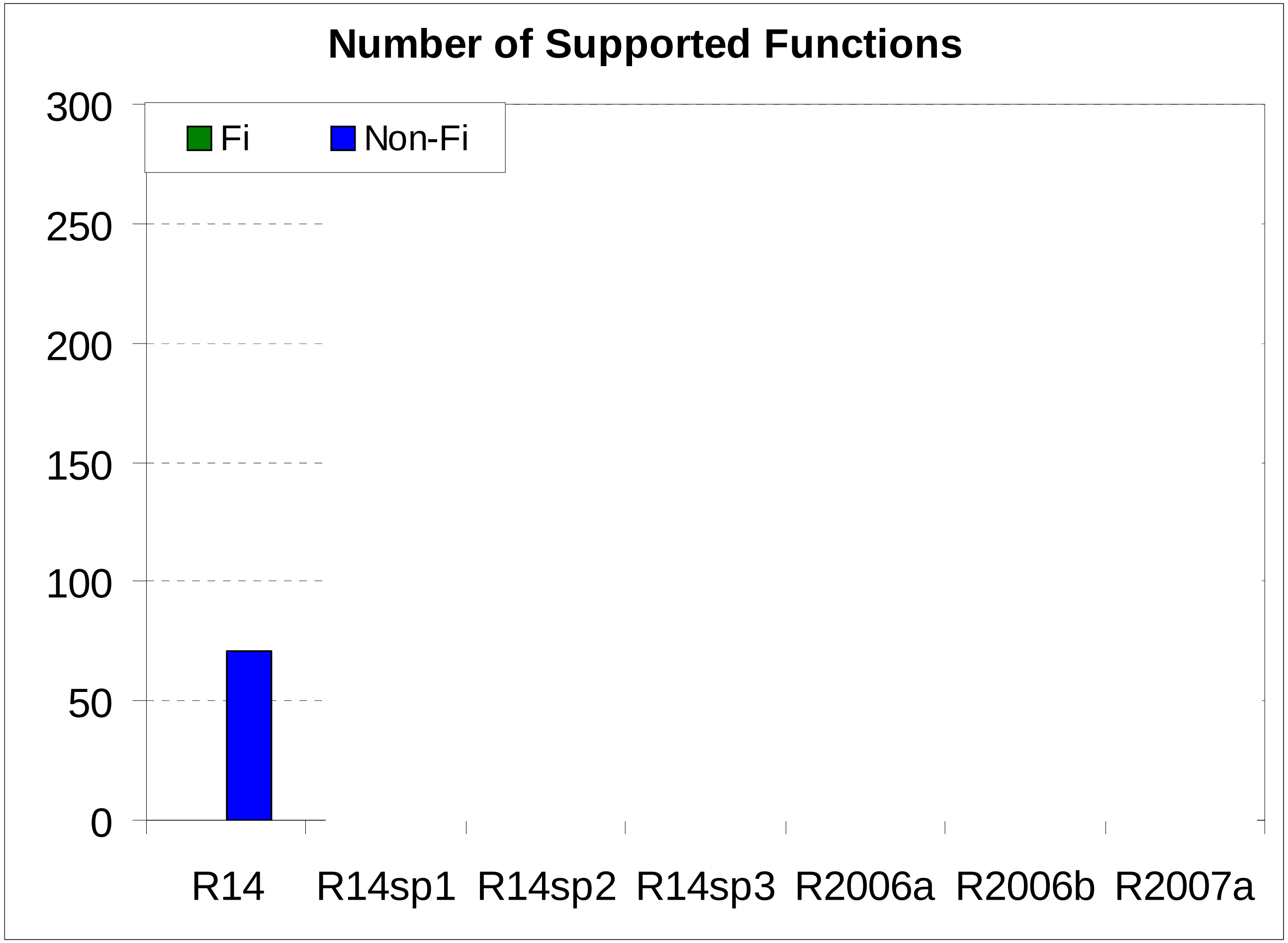
Historical Review

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

New features

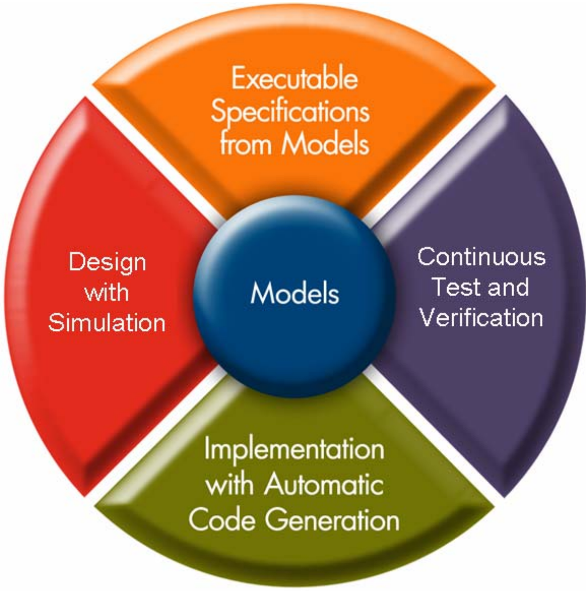
- Executable Specification
- Detailed Design
- Code Generation
- Code Integration

Embedded MATLAB Functions - 2007a



>>eml_frames, eml_nd

Agenda



Historical Review

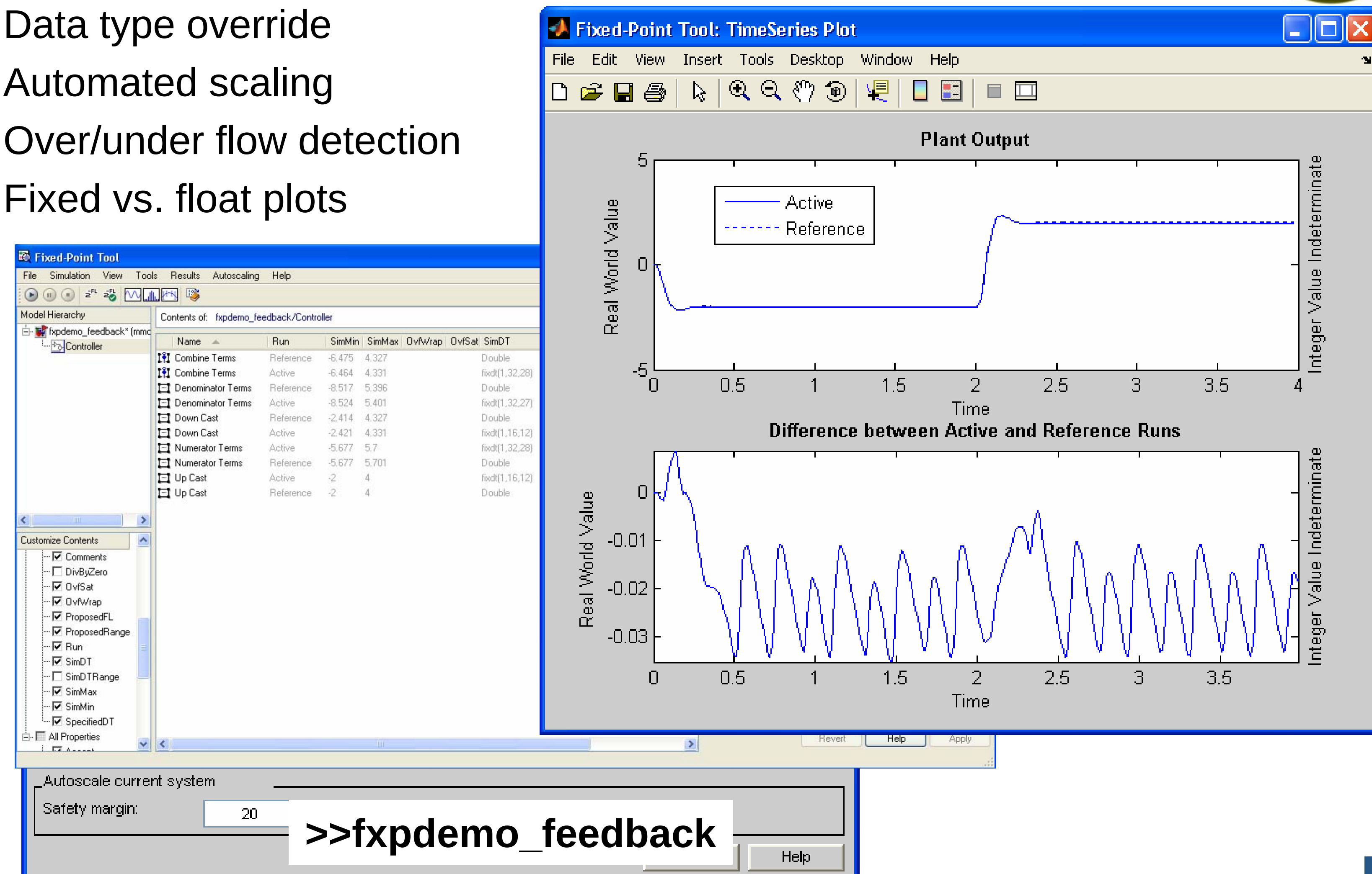
- Code Generation – 1999 (Release 11)
- Code Generation – 2005 (Release 14)
- Code Generation and Verification – R2007a

New features

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

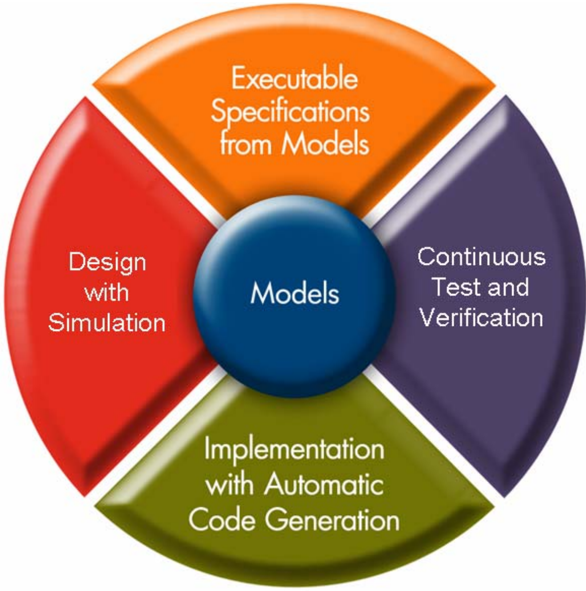
Fixed Point Tool – 2007a

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



>>fxpdemo_feedback

Agenda



Historical Review

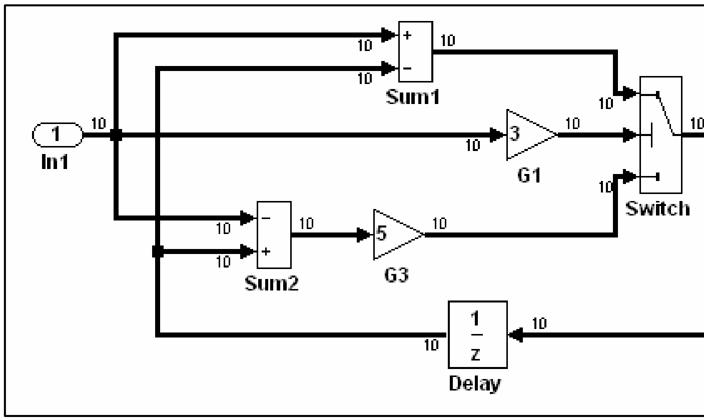
- Code Generation – 1999 (Release 11)
- Code Generation – 2005 (Release 14)
- Code Generation and Verification – R2007a

New features

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

Code Efficiency - R2007a

Wide Signals (for-loops)



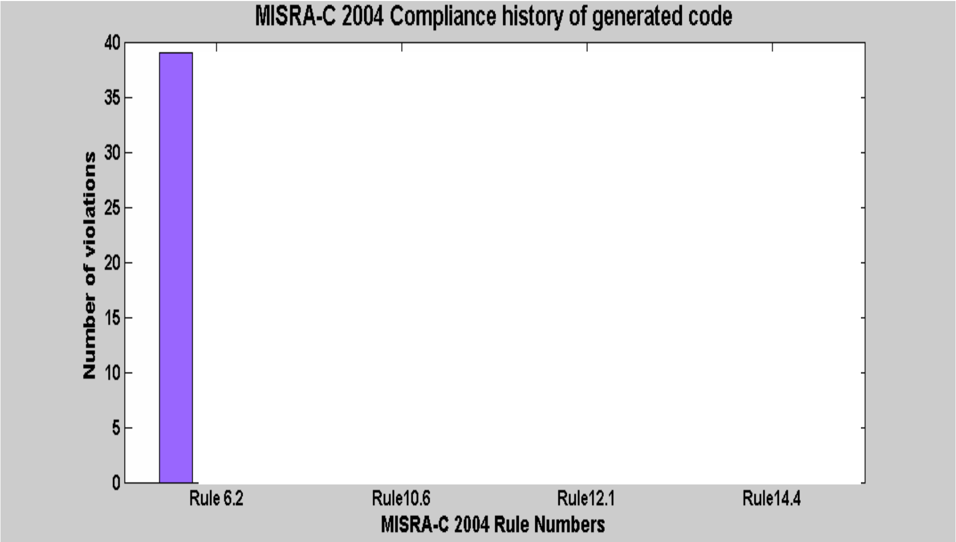

```
wdemo_forloop_R13SP2_step(void)
'rtb_Sw real_T rtb_Switch
{
 int_T i1;

 ... y0 = &rtb_Switch[0];
 ...= 0.0) {
 ...DSTATE[i1]);
 ...] +
 ...DSTATE[i1]) * 5.0);
 ...1[0];
 }
 {
 int_T i1;

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

 for ...=0; i1 < 10; i1++) {
 ..._DSTATE[i1] = u0[i1];
```

```
void rtwdemo_forloop_R14SP2_step(void)
{
 real_T rtb_S real_T rtb_Switch[10]

 {
  int32_T i1;

  for ; i1<10; i1++) {
   ...emo_forloop_R14SP2_U.In1[i1] * 3.0 >= 0.0) {
    rtb_Switch[i1] = rtwdemo_forloop_R14SP2_U.In1[i1] -
     rtwdemo_forloop_R14SP2_DWork.Delay_DSTATE[i1];
   } else {
    rtb_Switch[i1] = (rtwdemo_forloop_R14SP2_DWork.Delay_DSTATE[i1] -
     rtwdemo_forloop_R14SP2_U.In1[i1]) * 5.0;
   ...
   ...rloop_R14SP2_Y.Out1[i1] = rtb_Switch[i1];
   ...rloop_R14SP2_DWork.Delay_DSTATE[i1] = rtb_Switch[i1];
```

```
void rtwdemo_forloop_R2007a_step(void)
{
  real_T tmp
  for                    ++) {
                         ] >= 0.0) {
                         - rtDWork.Delay_DSTATE[i];
    } else {
      tmp = (rtDWork.Delay_DSTATE[i] - rtU.In1[i]) * 5.0;
    }

    rtY.Out1[i] = tmp;
    rtDWork.Delay_DSTATE[i] = tmp;
  }
 }
}
```

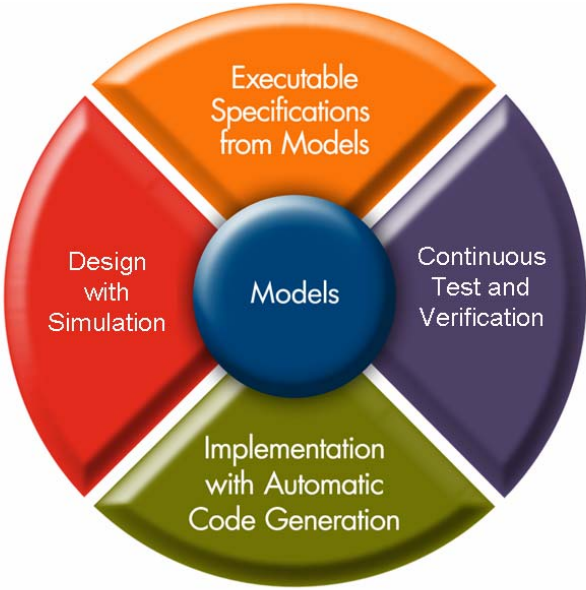
MISRA-C 2004 Support – R2007a

- 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



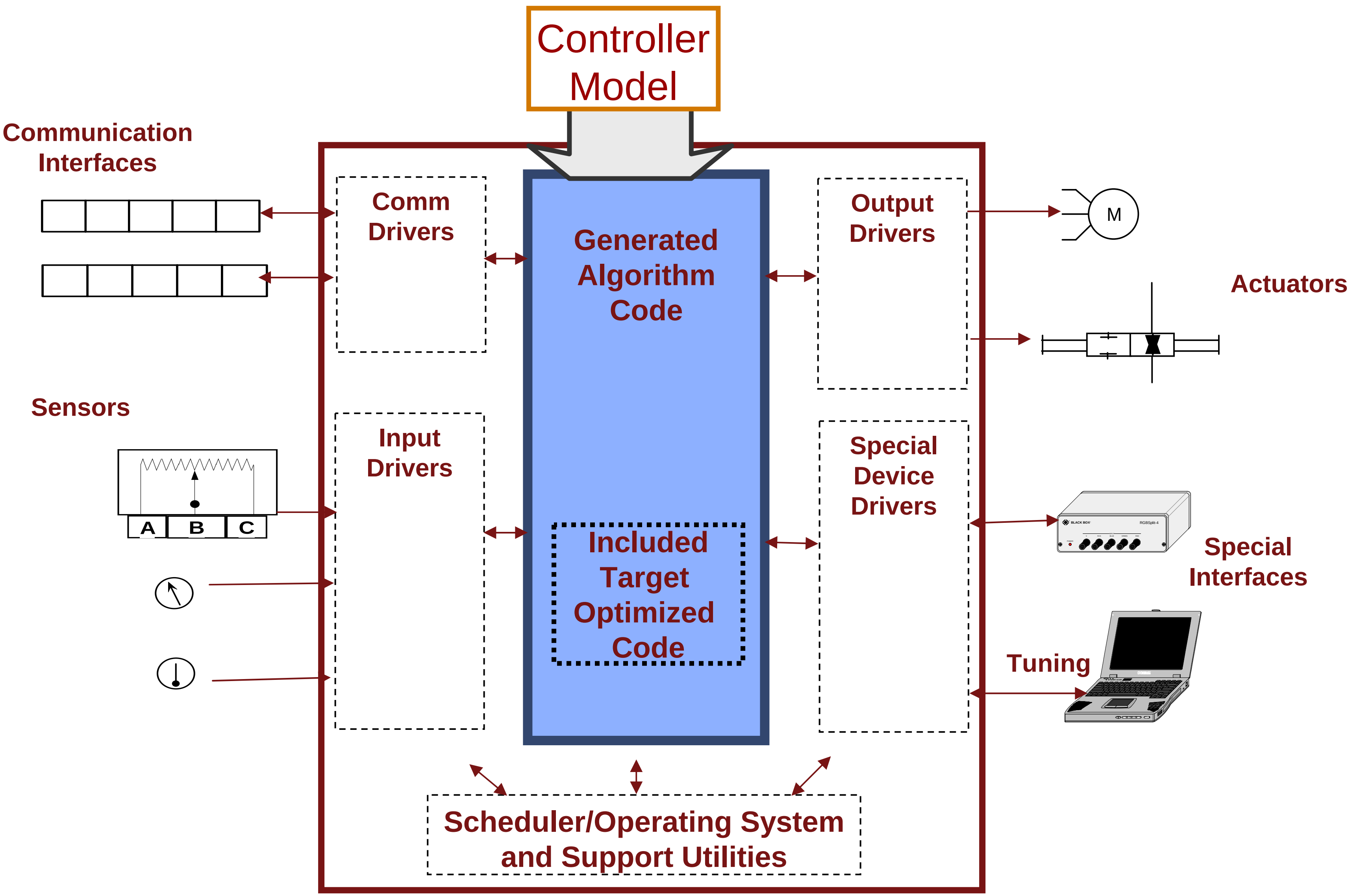
Historical Review

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

New features

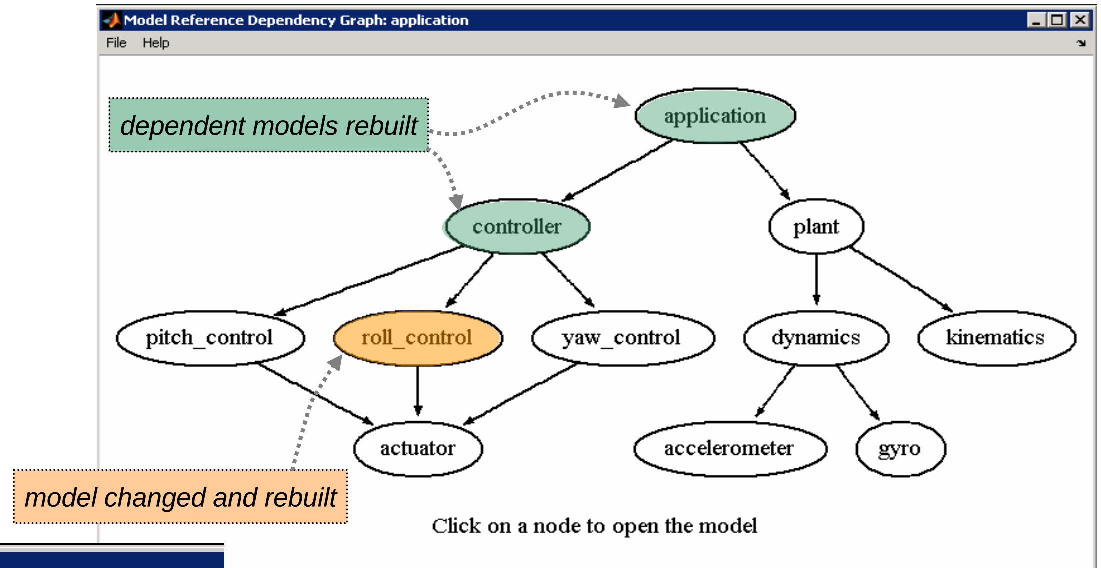
- Executable Specification
- Detailed Design
- Code Generation
- Code Integration

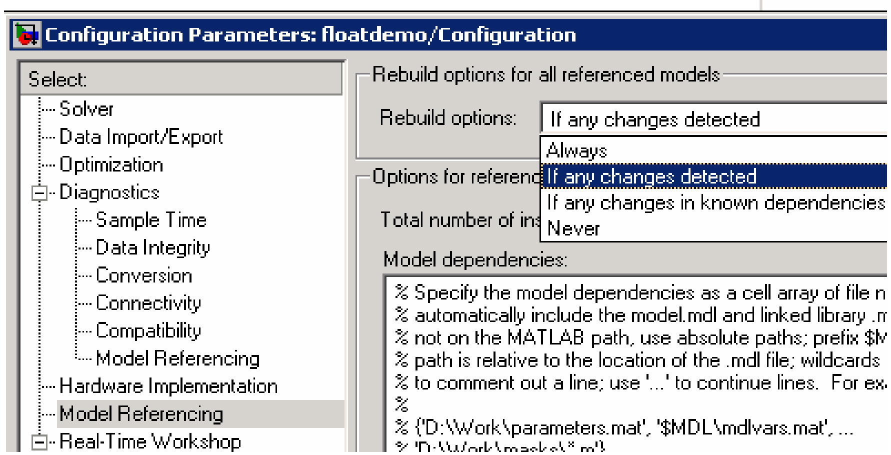
Code Architecture - Integration



Model Integration via Model Reference – R14

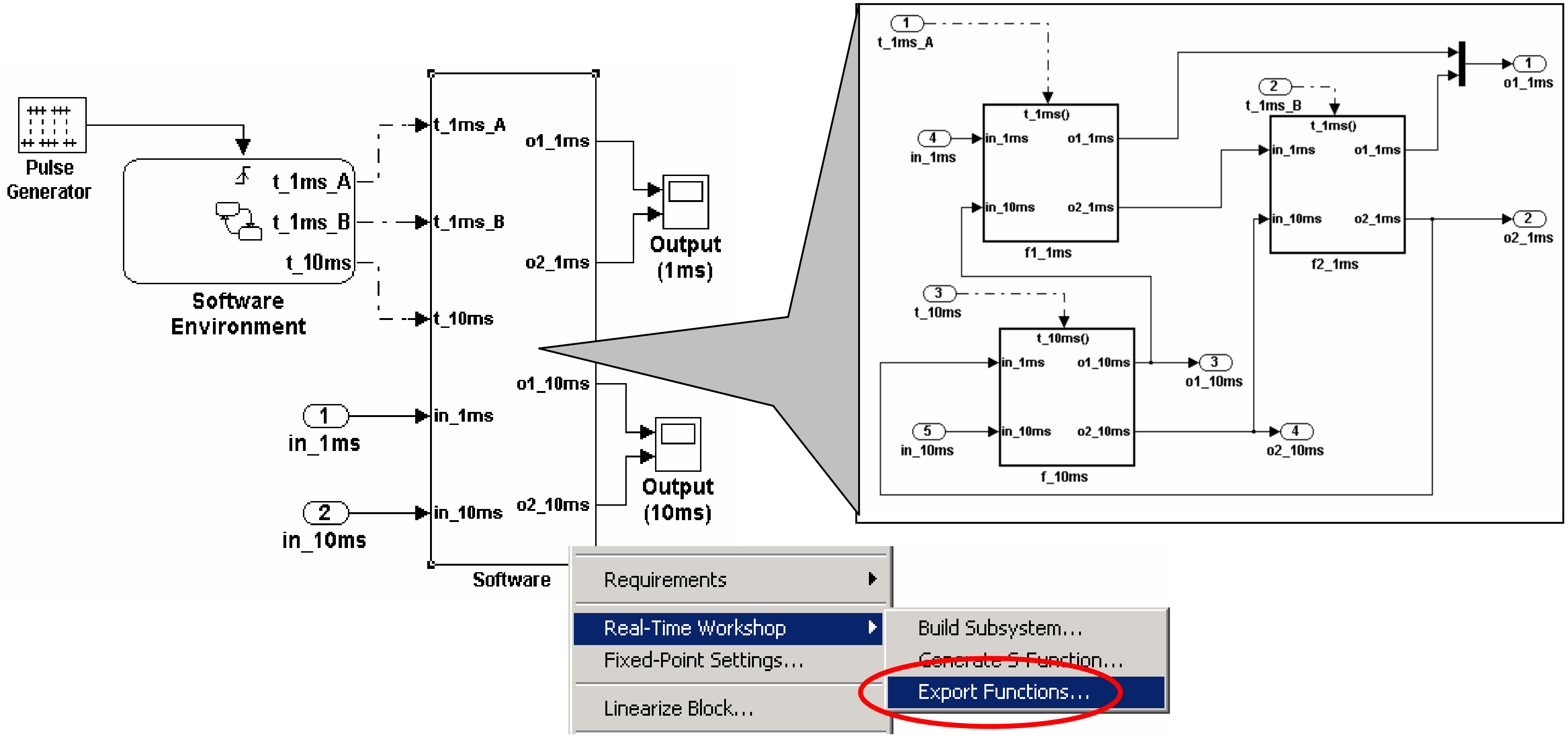
- Incremental code generation is supported via Model Reference





>> rtwdemo_mdlreftop
>> sldemo_mdlref_depgraph

Subsystem Integration w/Export Functions – R2006a



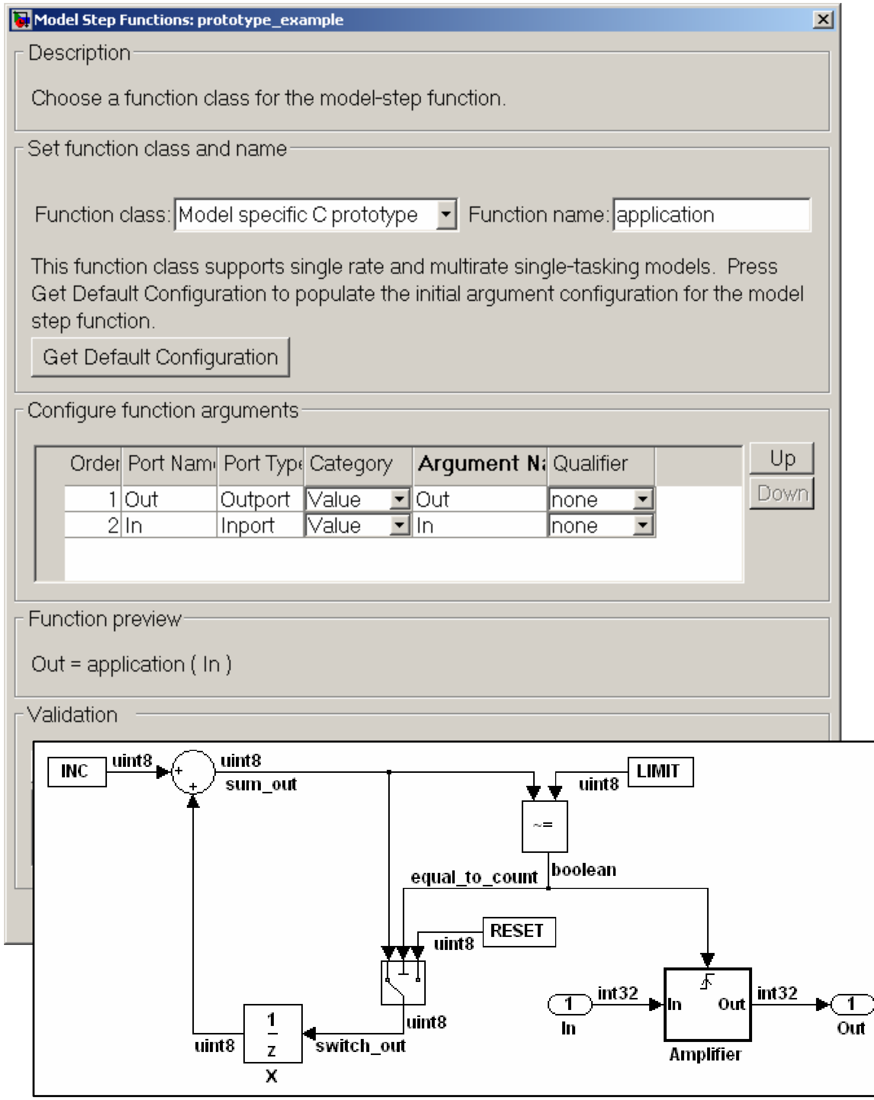
- **Supports a popular scheduling technique in production**
- **Streamlines code generated**
  - **No scheduler, no model step function**

>> rtwdemo_export_functions

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

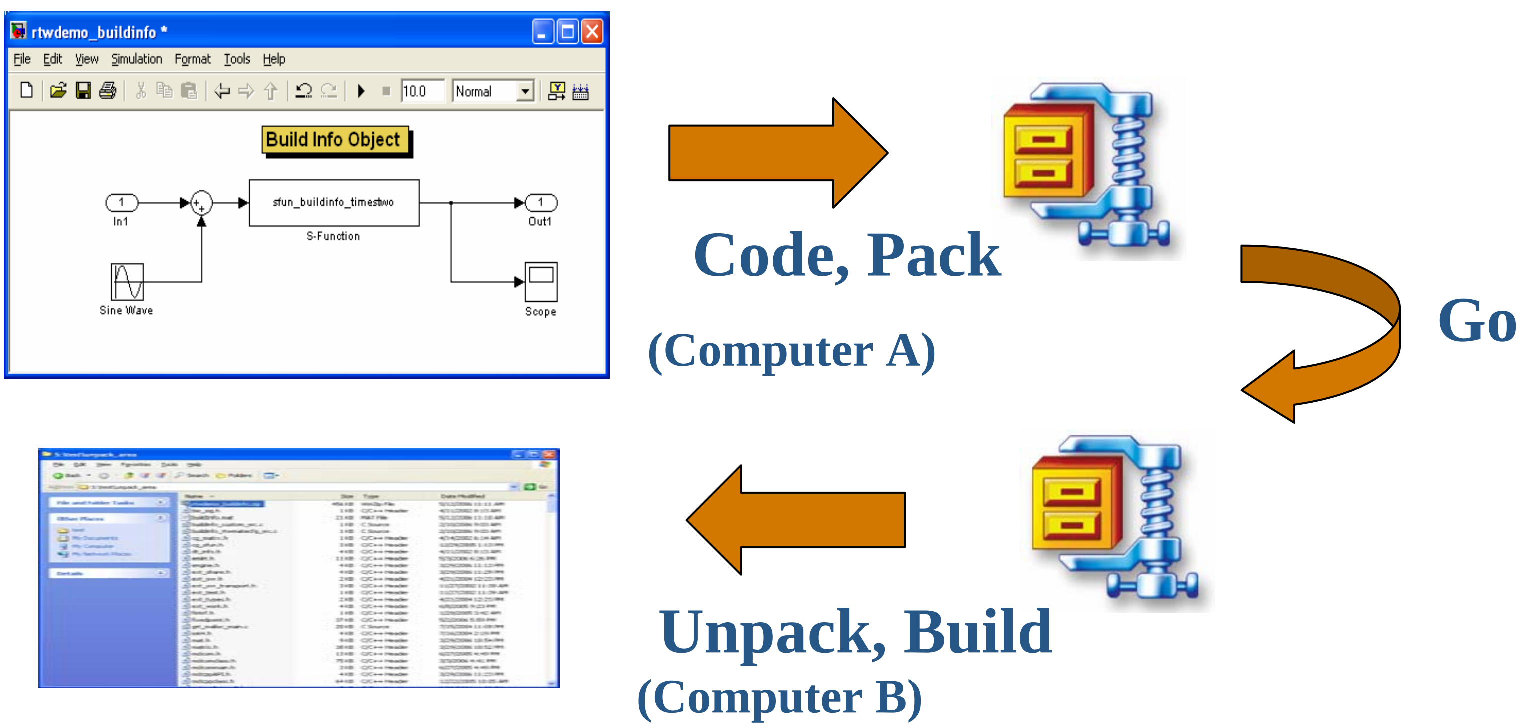
Production Code Integration Comparison

	Model Reference Integration	Model Step Function Integration	Subsystem (Function Export) Integration
Fidelity / Control	LOW-MID	MID	HIGH
Convenience	Fully automated, incremental code HIGH	F[illegible] control	Manual integration LOW
Applications	Large Scale Mode[illegible] System Focu[illegible]	Model Bound[illegible] System or Softwa[illegible]	Software Engineering Focused

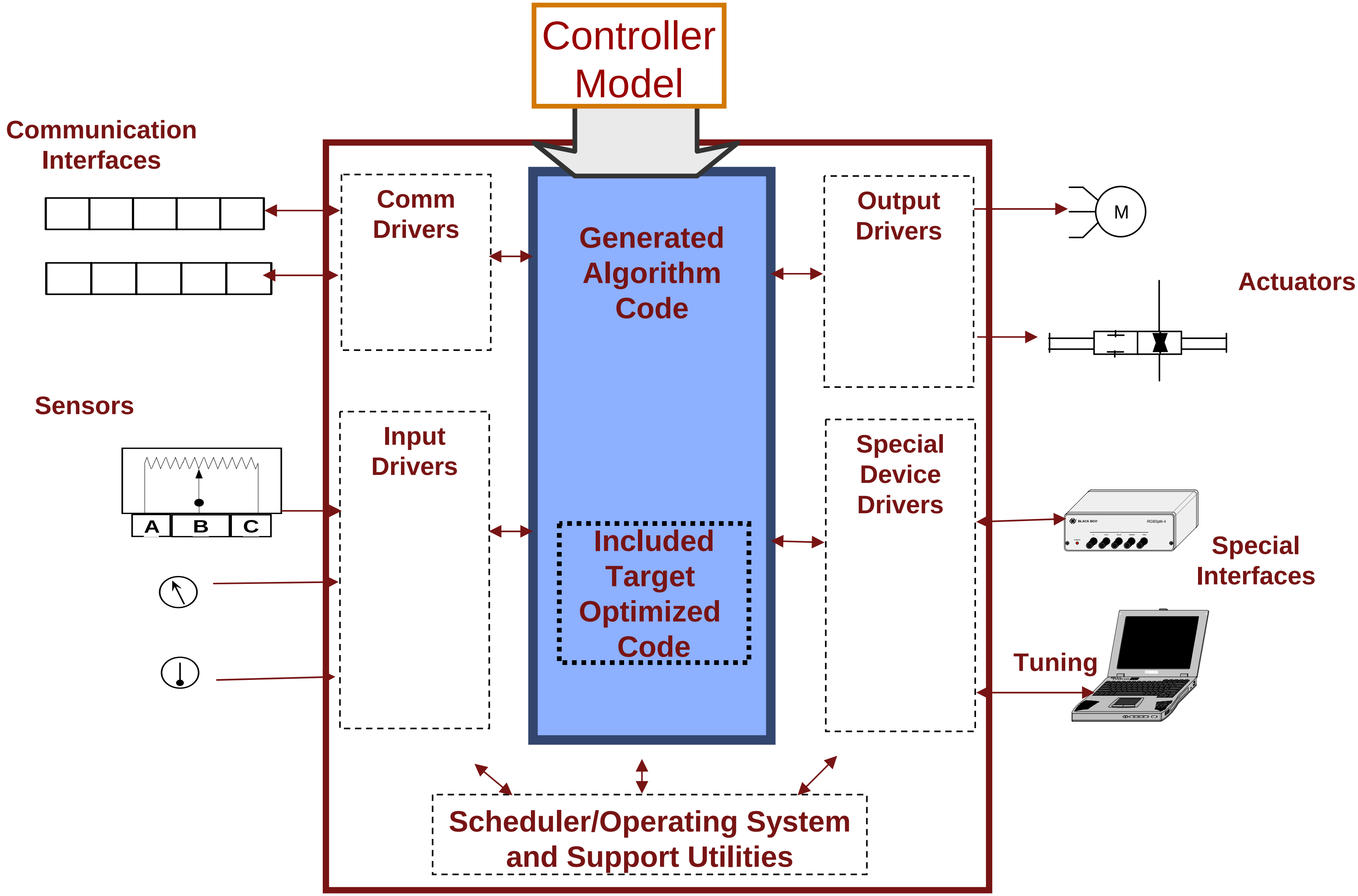
Convenience

Pack-and-Go for Relocating Generated Code – R2006b

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



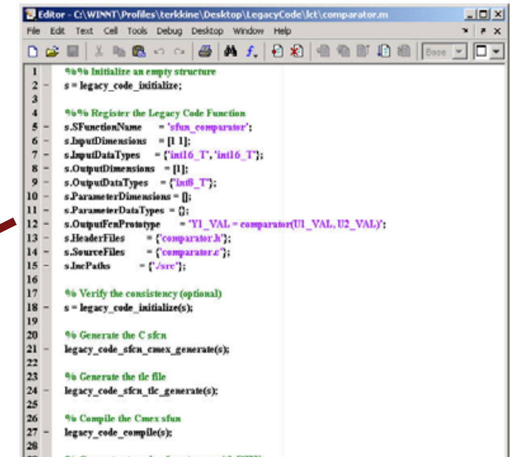
Code Architecture - Integration



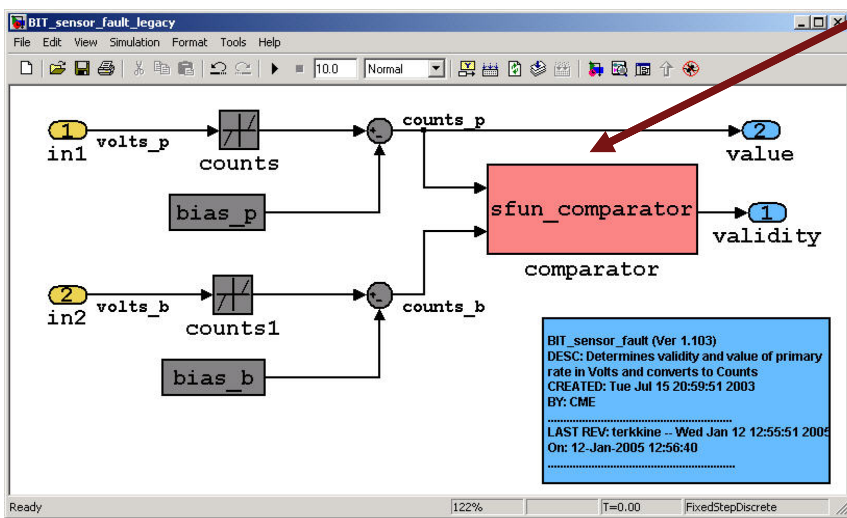
Legacy Code Tool – R2006b

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

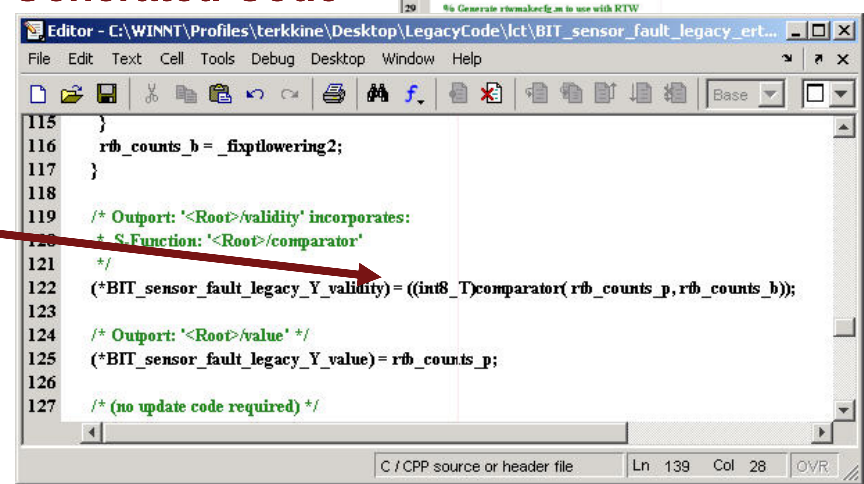
Registration File



Simulation Model



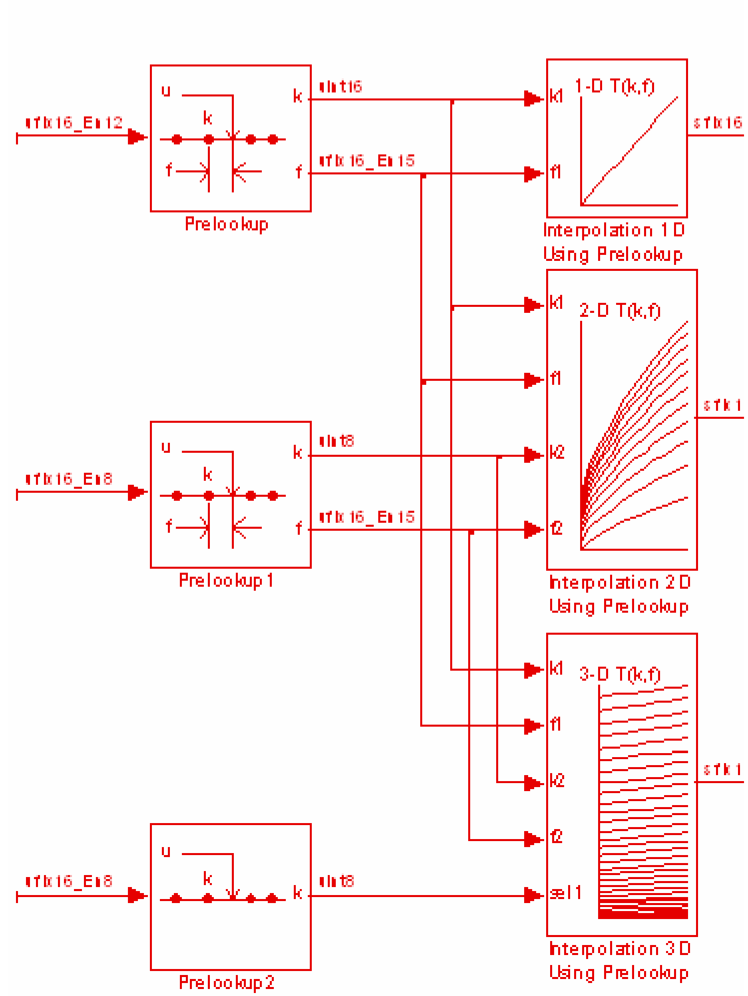
Generated Code



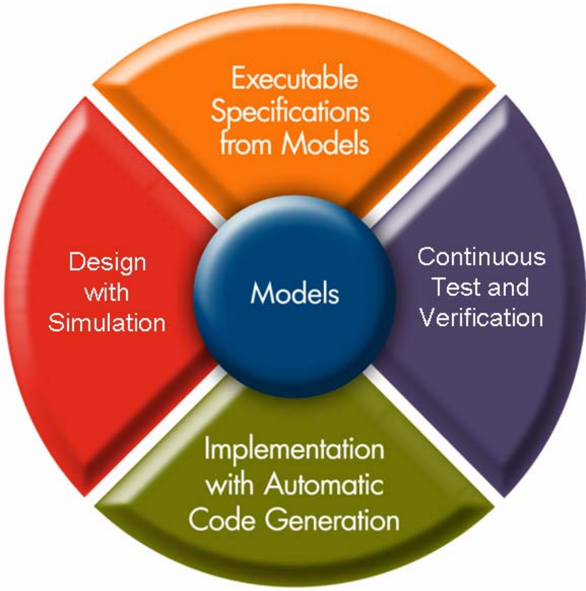
`>> sldemo_lct_lut`

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

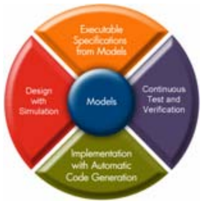
- Code Generation – 1999 (Release 11)
- Code Generation – 2005 (Release 14)
- Code Generation and Verification – R2007a

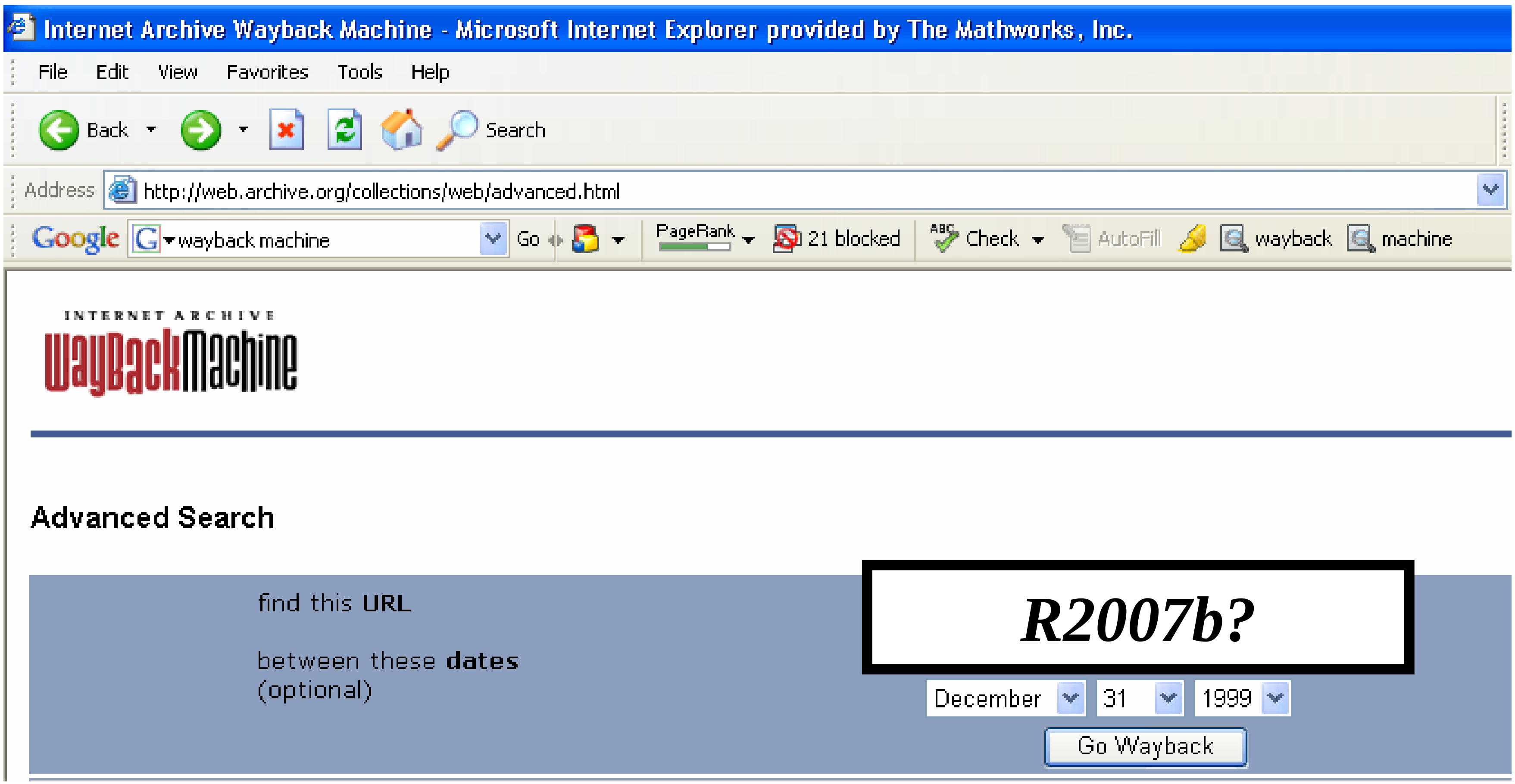
New Features

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

The Future?

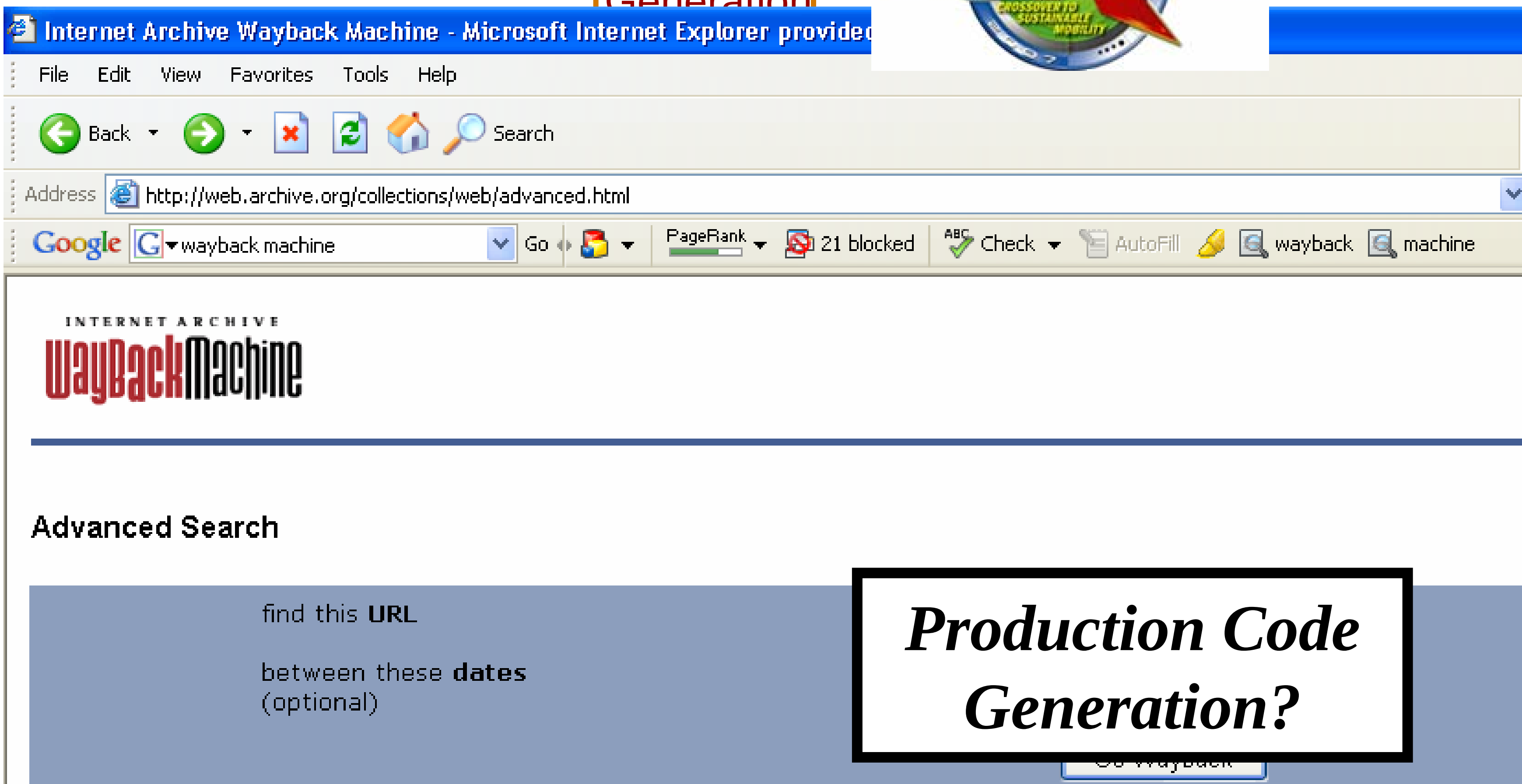
Way Forward Machine





Way Forward Machine

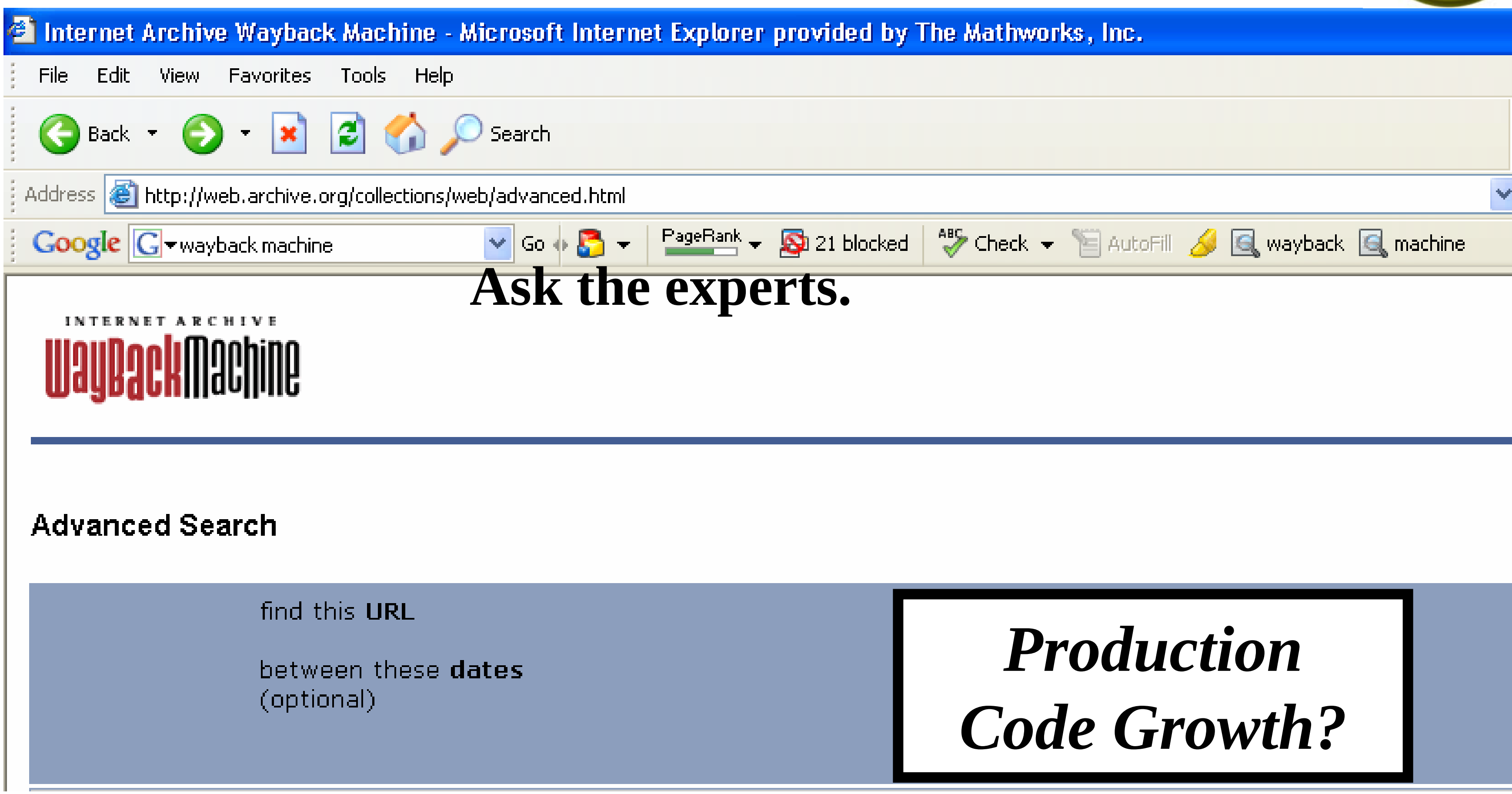
Application Generation



Production Code Generation?

Scheduler/Operating System and Support Utilities

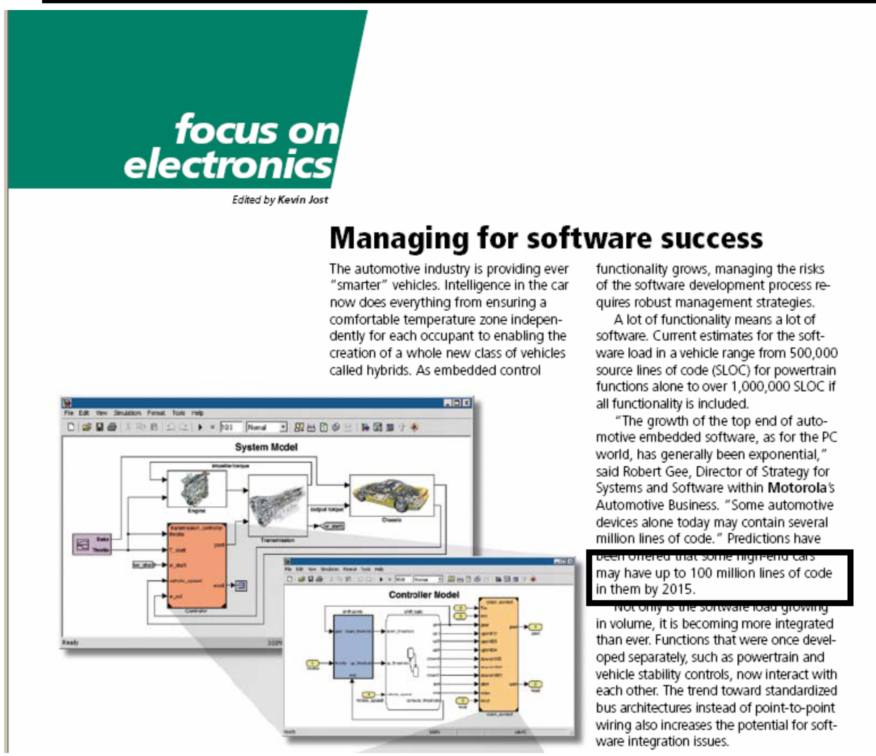
Way Forward Machine



Exponential Growth of Embedded Code

Estimated Source Lines of Code (LOC)

- Today's powertrain: 500,000 LOC
- Today's vehicles: 1,000,000 LOC



focus on electronics

Edited by Kevin Jost

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." Predictions have been offered that some high-end cars may have up to 100 million lines of code in them by 2015.

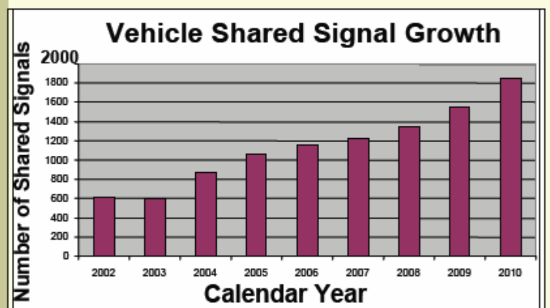
Not only is the software load growing in volume, it is becoming more integrated than ever. Functions that were once developed separately, such as powertrain and vehicle stability controls, now interact with each other. The trend toward standardized bus architectures instead of point-to-point wiring also increases the potential for software integration issues.

> **"Growth of top end automotive embedded software has been exponential."**
>
> **Robert Gee**
>
> **Director of Strategy for Motorola Automotive**

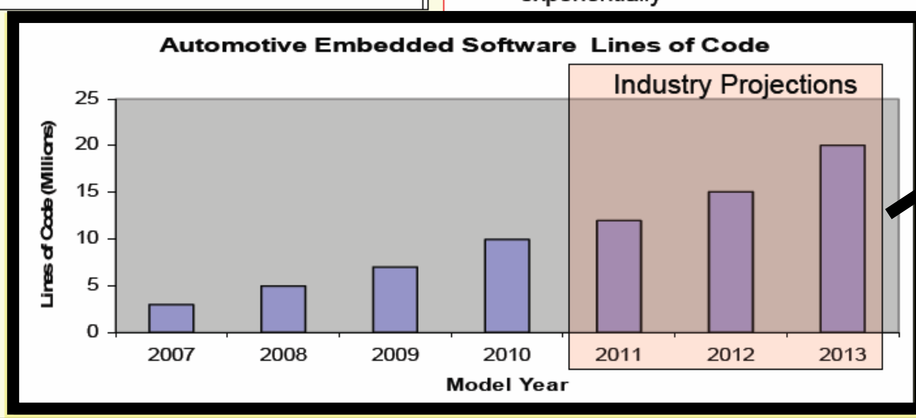
Automotive Engineering,
"Managing for Software Success" – Aug 2006
sae.org/automag/electronics/08-2006/1-114-8-34.pdf

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

ICSE May 2007

Page 4

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.

Automotive Engineering,
"Managing for Software Success" – Aug 2006
sae.org/automag/electronics/08-2006/1-114-8-34.pdf

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-constra
show how you can use MathWorks products to:

- Establish a software engineering environment that focuses on developing a detailed software
- 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

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 start 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

Web Browser - Module 0: Introduction

File Edit View Go Debug Desktop Window Help

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

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:

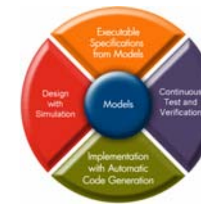
- www.mathworks.com/aboutus/contact_us

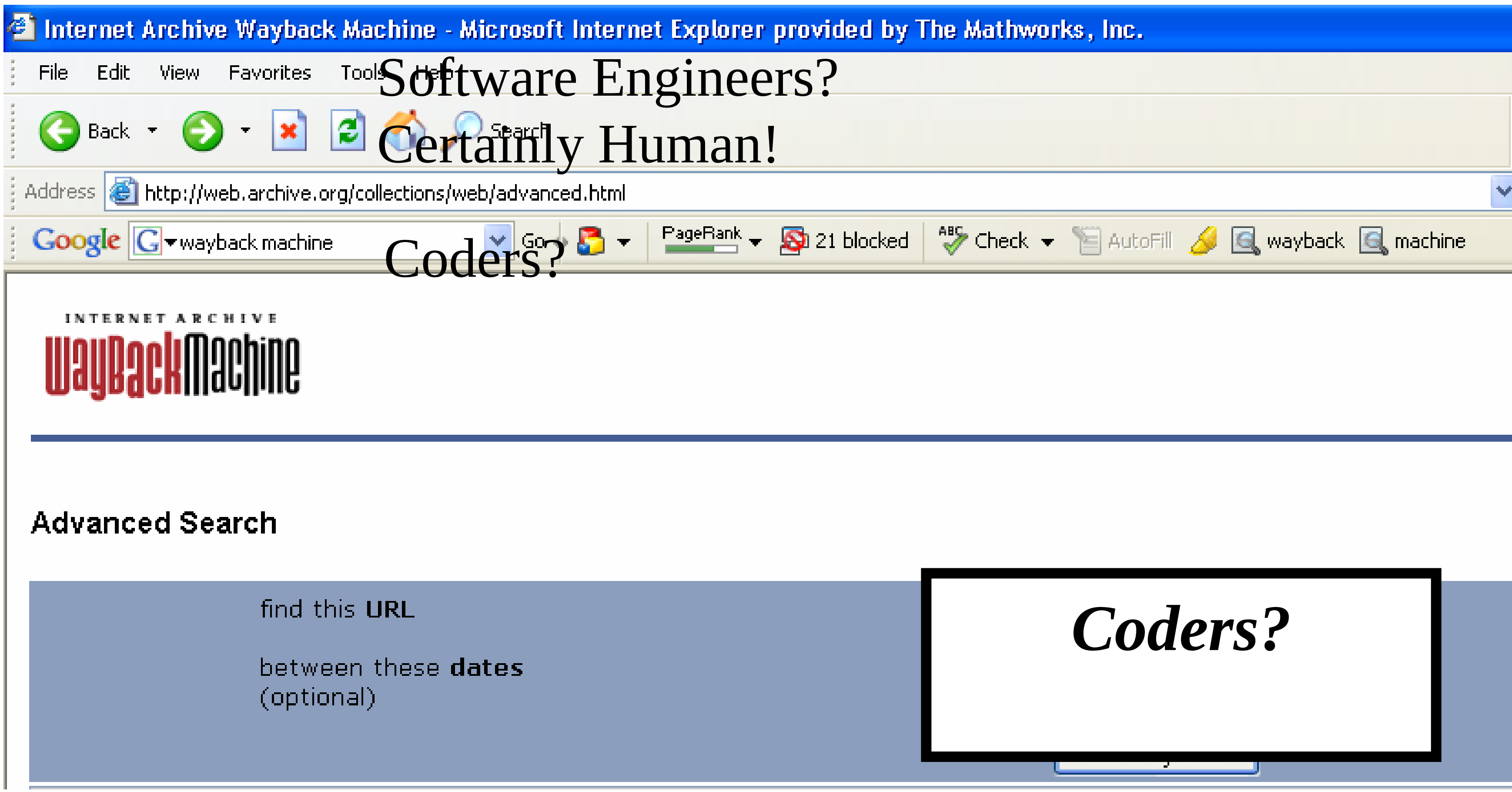
We are very interested in learning about your production code needs.

Thank you – tom.erkkinen@mathworks.com

One More Thing …

Way Forward Machine (www.archive.org)





Software Engineers?

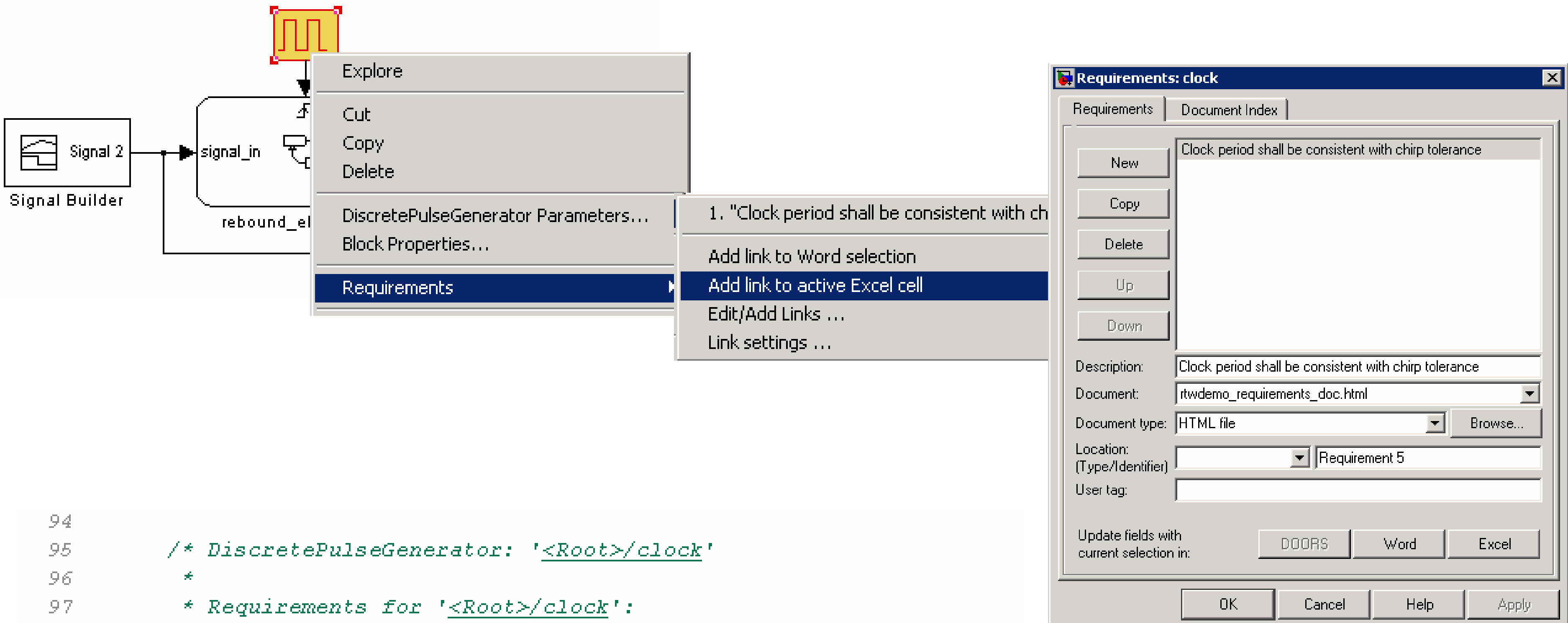
Certainly Human!

Coders?

Coders?

-- Reference Material --

Add Links to Requirements




```
/* 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) {
```

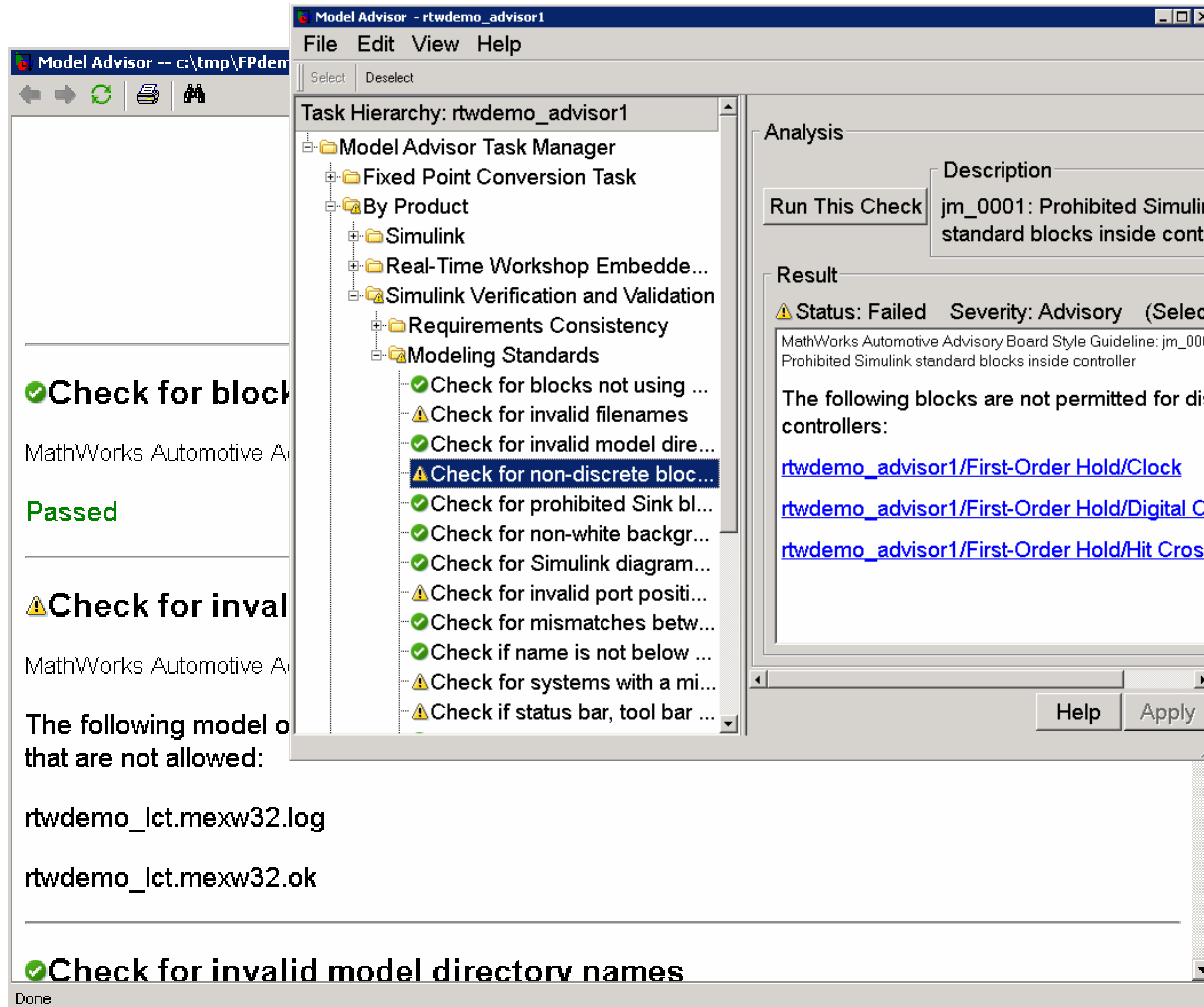
Requirements appear in the code

>> rtwdemo_requirements

Enforce Modeling Standards

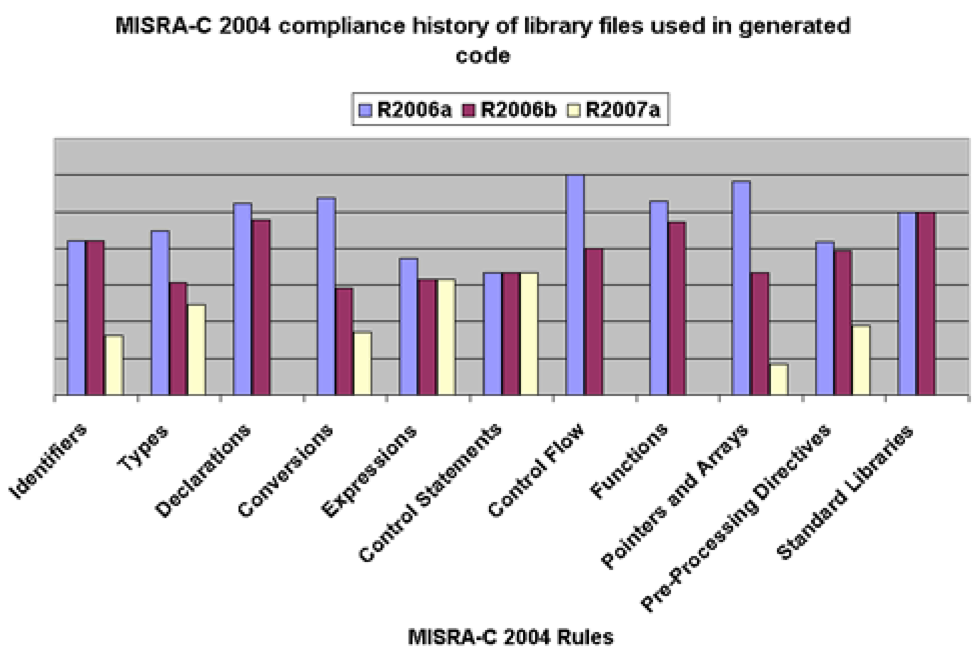
Simulink Model Advisor

- 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/libsrc 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**

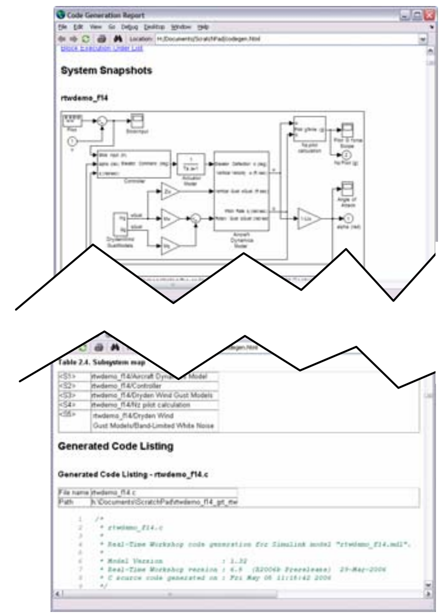


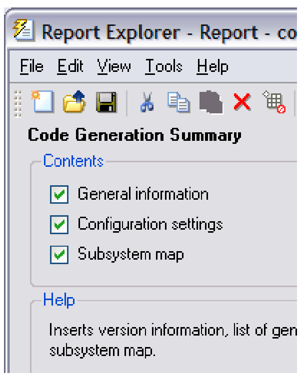
Documentation of the Generated Code

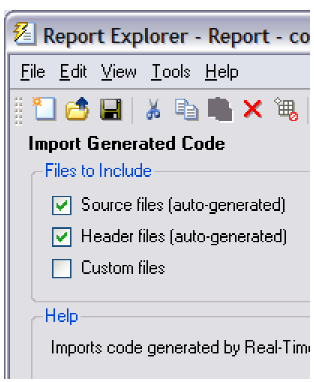
	HTML report	Code node	Pre-canned report	Report generator
Need setup	No	No	No	Yes
Customizable	No	No	No	Yes
Link to model objects	Yes	Yes	No	No
Portable document	Yes*	No	Yes	Yes
Containing model snapshots	No	No	Yes	Yes
Multiple file formats	No	No	No	Yes
Common use case	Interactive review	Interactive review	Quick deposition	Formal documentation
Require Report Generator	No	No	Installation Required	Installation and license Required

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

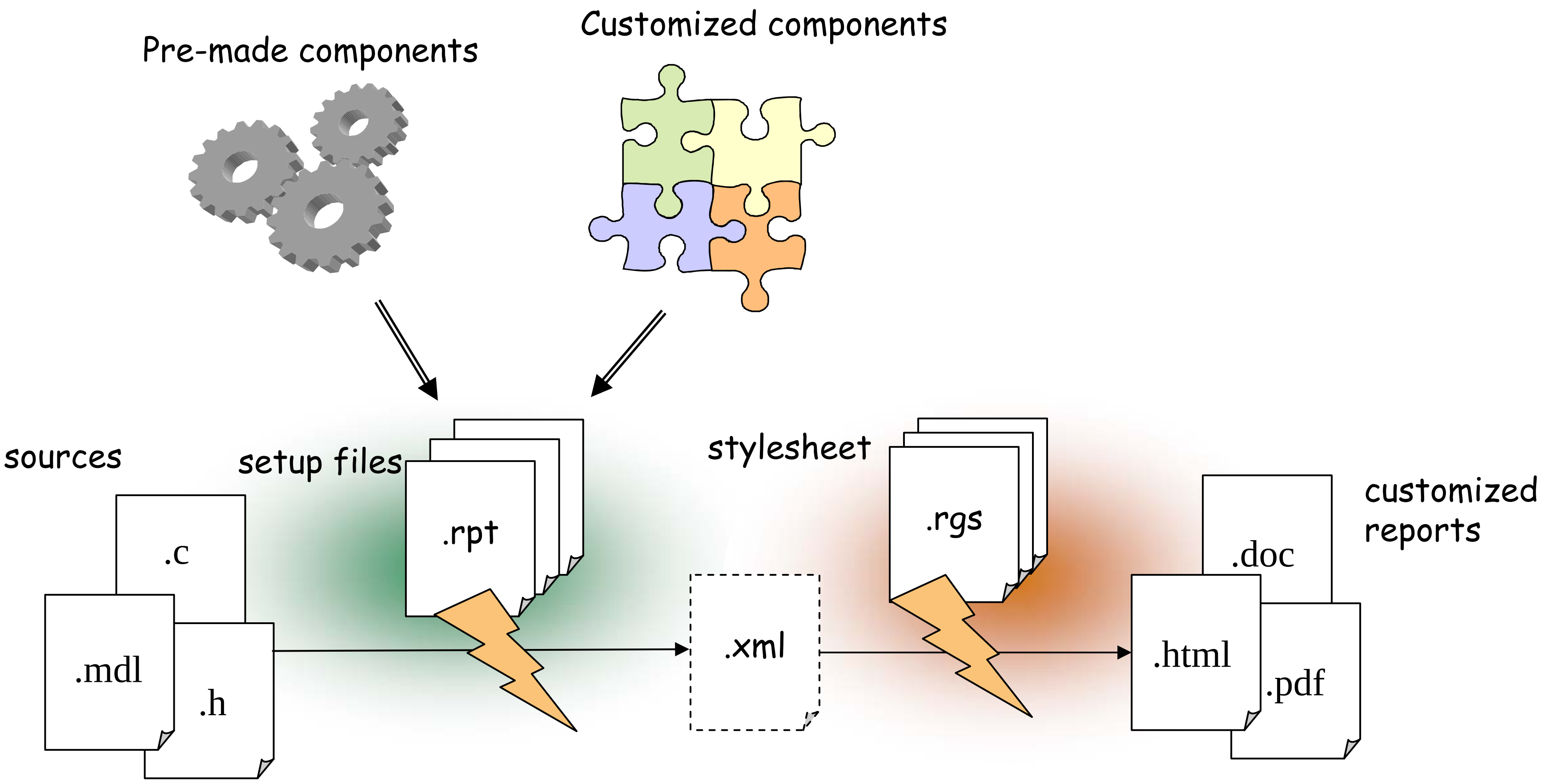







```
>> rtwdemo_codegenrpt
>> help rtwReport
```

Creating Customized Report



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)

Editor - D:\Work\workarea\R2006atemp\buildinfo_demo.m*

```
function buildInfo_demo(buildInfo)

% Write all generated files to an HTML report
fid = fopen('BuildInfoReport.html','wt');
fprintf(fid,'<html>\n');
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>');
for i = 1:length(allSrcFileList)
    fprintf(fid,'%s <br>', allSrcFileList{i});
end

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

buildInfo_demo | Ln 12 Col 60 | OVR

buildinfo_demo.m

Build Info API

Location: ert_rtw/BuildInfoReport.html

Build Info API Demo

Generated Source and Header Files:

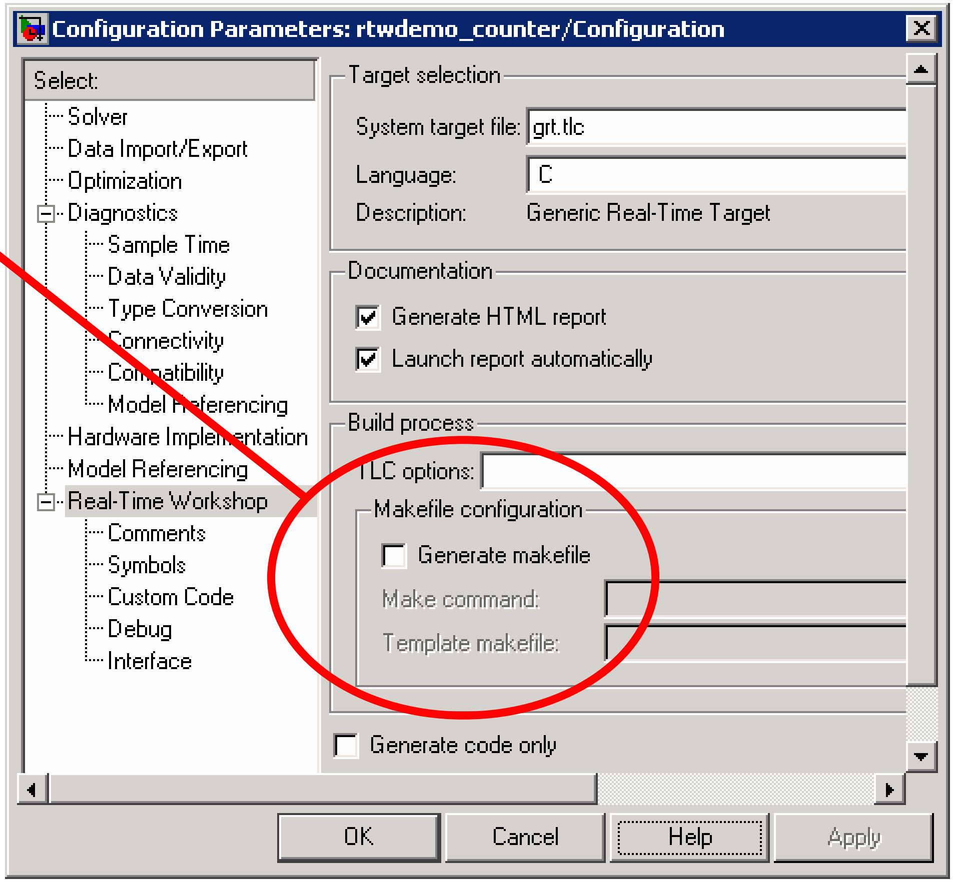
rtwdemo_counter.h
rtwdemo_counter_private.h
rtwdemo_counter_types.h
rtwtypes.h
ert_main.c
rtwdemo_counter.c

BuildInfoReport.html

>> 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



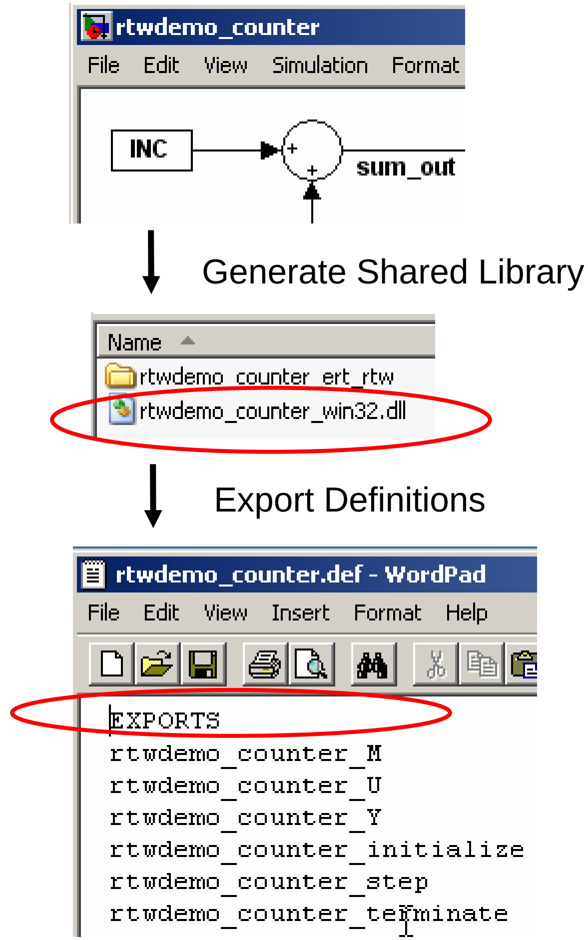
Shared Library Target Solution

Problem

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

Solution

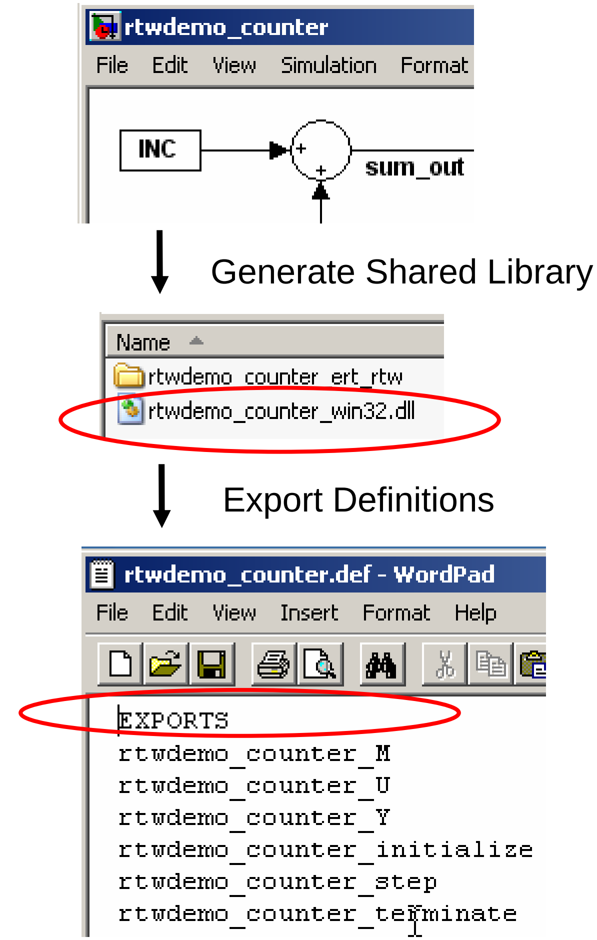
- RTW-EC shared library target
- Generate a shared library (.dll or .so) version of the generated code with documented interface
- Support initialize, step, and terminate a simulation
- Unix and PC support



Shared Library Target Benefits

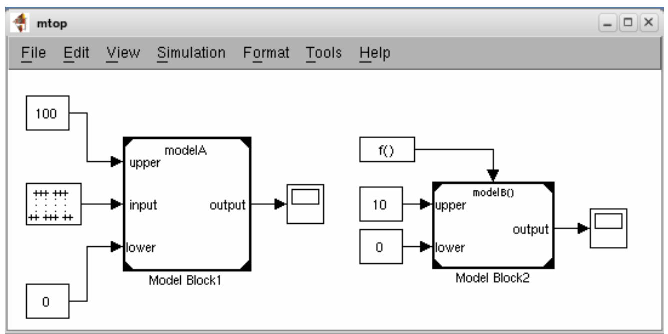
Benefit

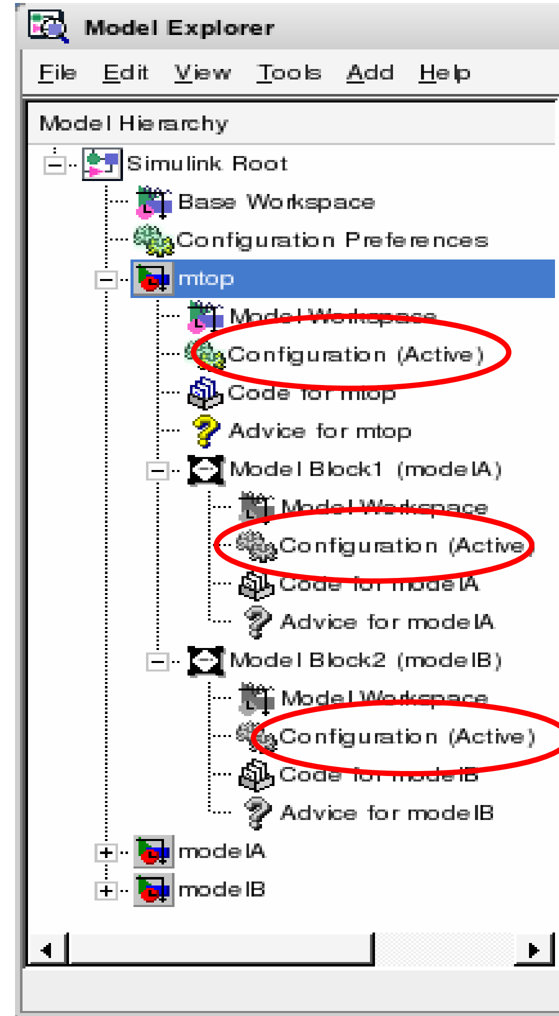
- Easier distribution of model simulation
- Shared library is reusable and upgradeable without recompiling the overall application
- Package and secure intellectual property



Configuring a Model Reference Hierarchy

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





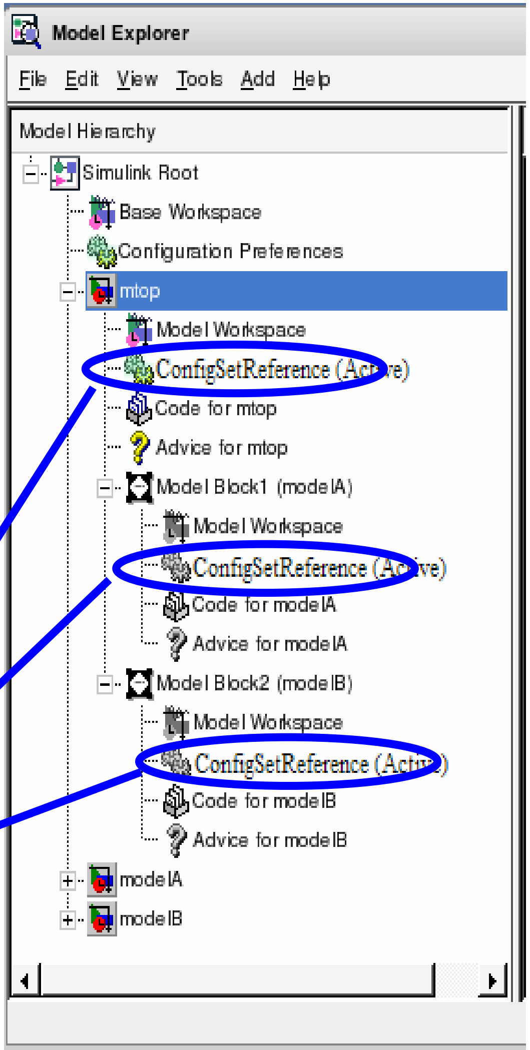
<u>Problem:</u>

- 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

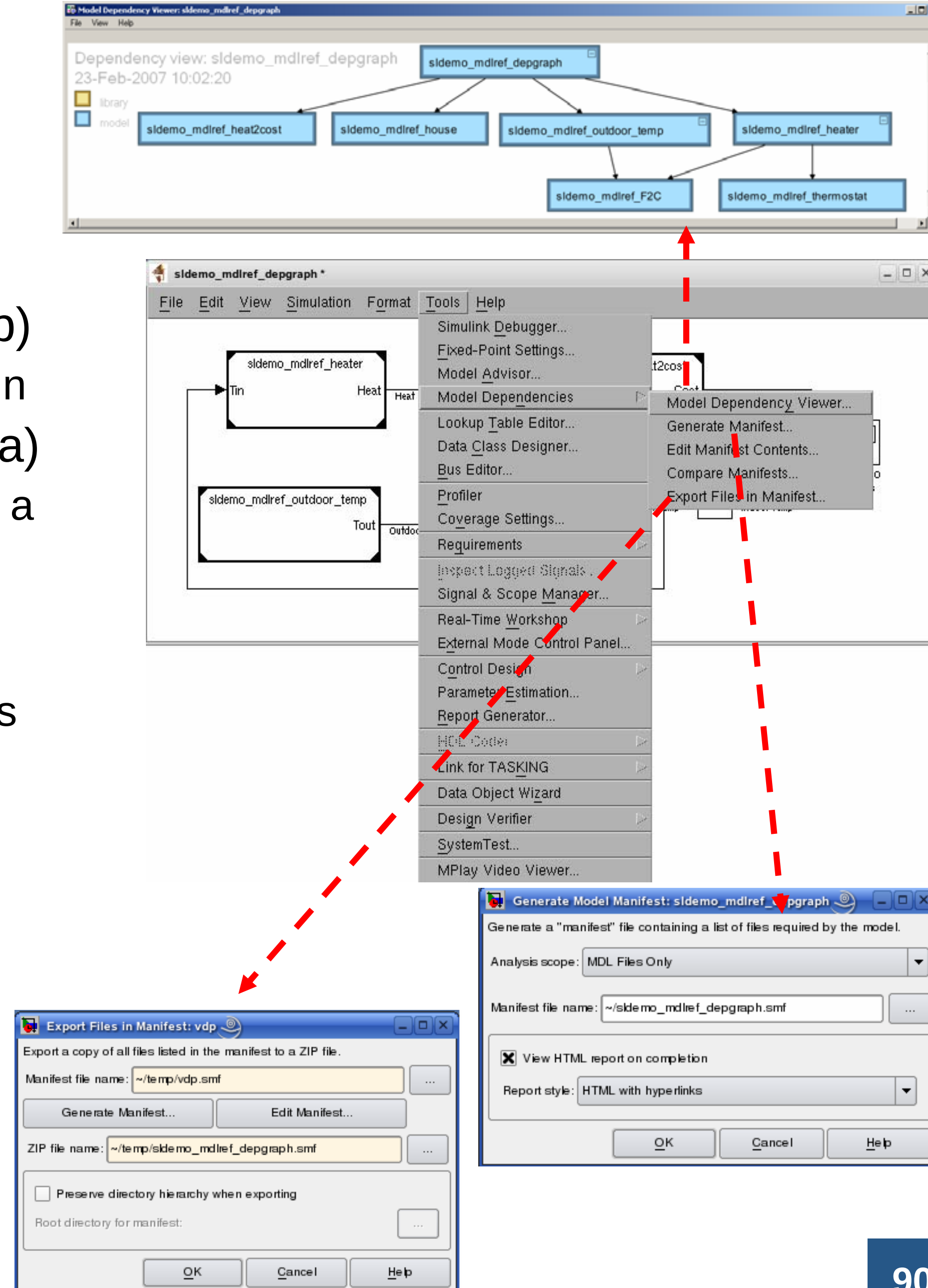
```
>> cs = Simulink.ConfigSet
or
>> cs = getActiveConfigSet(mdl);
```




```
>> 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