

# Latest Features in Embedded Coder

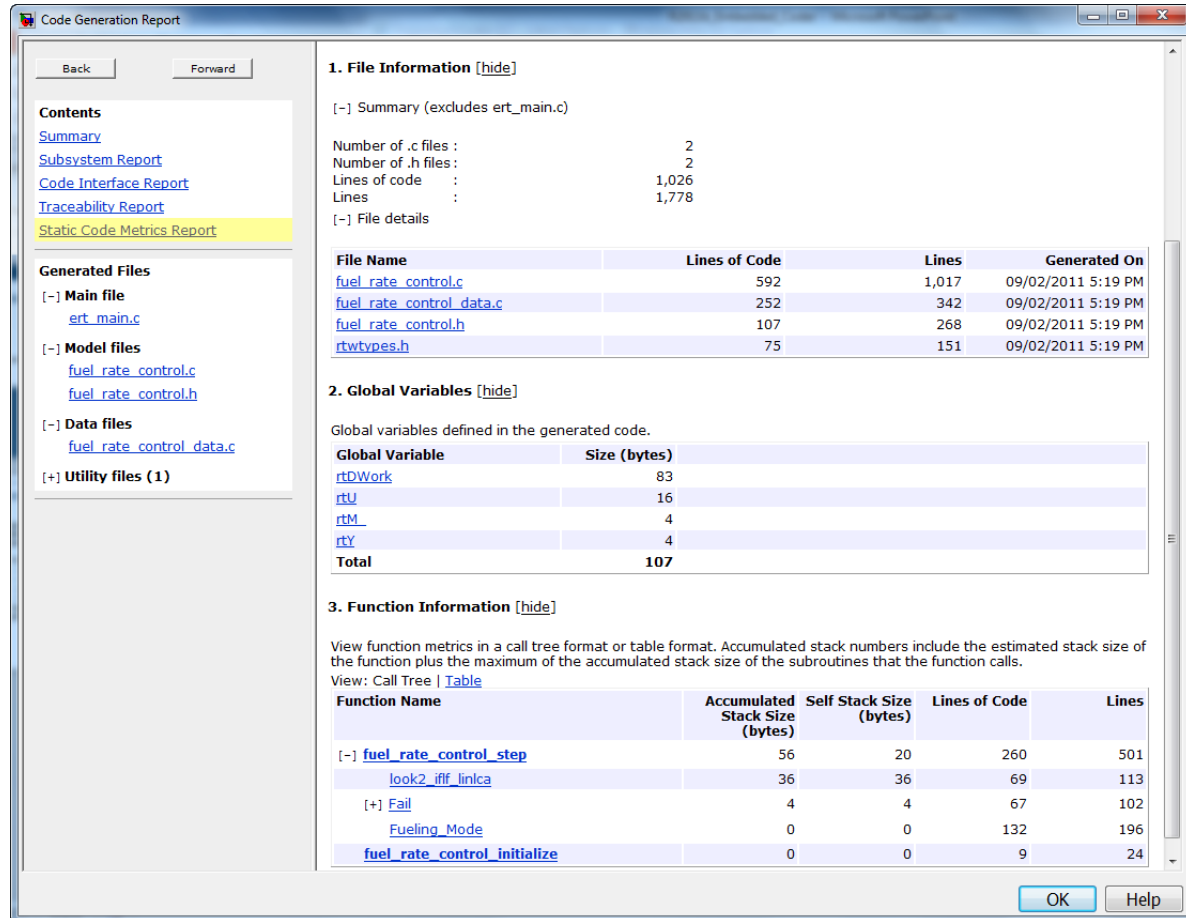
September 2011

**R2011b**

# Static Code Metrics Report

Assess if code is optimized for your embedded system

- Reports
  - Number of files
  - Number of lines
  - Global RAM
  - Stack size
- Static information, target independent
- Integration with code generation report



**Code Generation Report**

Back Forward

**Contents**

- [Summary](#)
- [Subsystem Report](#)
- [Code Interface Report](#)
- [Traceability Report](#)
- [Static Code Metrics Report](#)**

**Generated Files**

- [ - ] Main file
  - [ert\\_main.c](#)
- [ - ] Model files
  - [fuel\\_rate\\_control.c](#)
  - [fuel\\_rate\\_control.h](#)
- [ - ] Data files
  - [fuel\\_rate\\_control\\_data.c](#)
- [ + ] Utility files (1)

**1. File Information [hide]**

[ - ] Summary (excludes ert\_main.c)

Number of .c files : 2  
 Number of .h files : 2  
 Lines of code : 1,026  
 Lines : 1,778

[ - ] File details

File Name	Lines of Code	Lines	Generated On
<a href="#">fuel_rate_control.c</a>	592	1,017	09/02/2011 5:19 PM
<a href="#">fuel_rate_control_data.c</a>	252	342	09/02/2011 5:19 PM
<a href="#">fuel_rate_control.h</a>	107	268	09/02/2011 5:19 PM
<a href="#">rtwtypes.h</a>	75	151	09/02/2011 5:19 PM

**2. Global Variables [hide]**

Global variables defined in the generated code.

Global Variable	Size (bytes)
<a href="#">rtDWork</a>	83
<a href="#">rtU</a>	16
<a href="#">rtM</a>	4
<a href="#">rtY</a>	4
<b>Total</b>	<b>107</b>

**3. Function Information [hide]**

View function metrics in a call tree format or table format. Accumulated stack numbers include the estimated stack size of the function plus the maximum of the accumulated stack size of the subroutines that the function calls.

View: Call Tree | [Table](#)

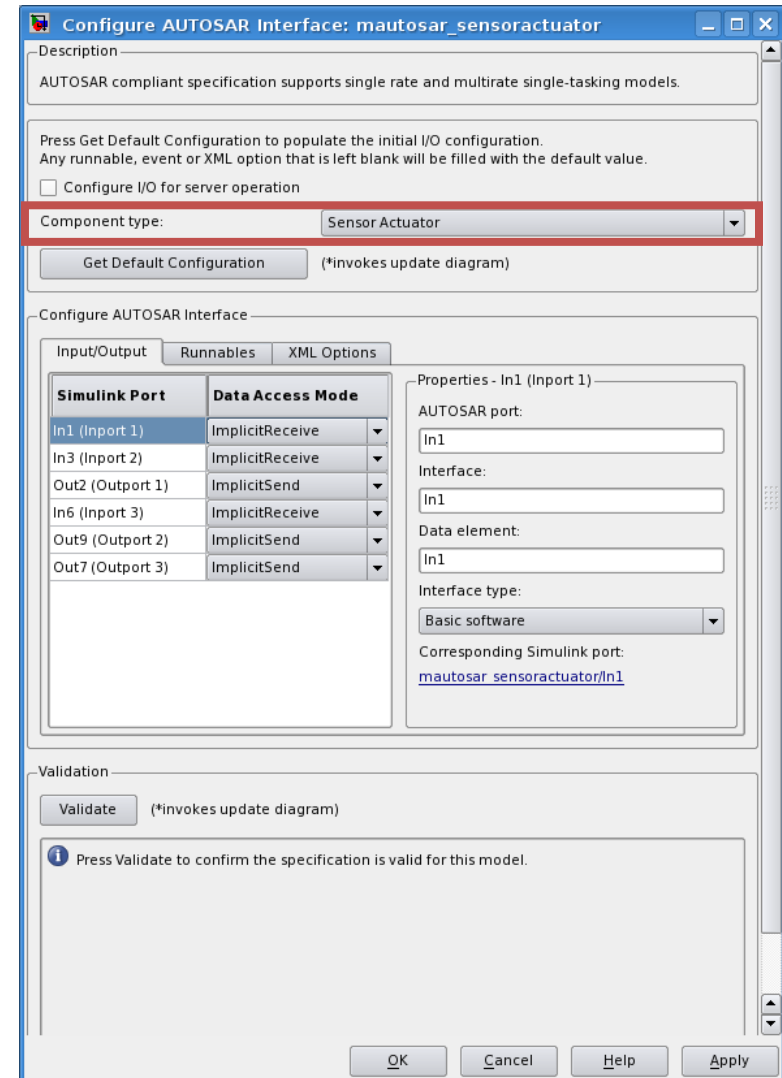
Function Name	Accumulated Stack Size (bytes)	Self Stack Size (bytes)	Lines of Code	Lines
[ - ] <a href="#">fuel_rate_control_step</a>	56	20	260	501
<a href="#">look2_iff_linca</a>	36	36	69	113
[ + ] <a href="#">Fail</a>	4	4	67	102
<a href="#">Fueling_Mode</a>	0	0	132	196
<a href="#">fuel_rate_control_initialize</a>	0	0	9	24

OK Help

# AUTOSAR Sensor/Actuator Support

## Design sophisticated AUTOSAR sensor/actuator components in Simulink

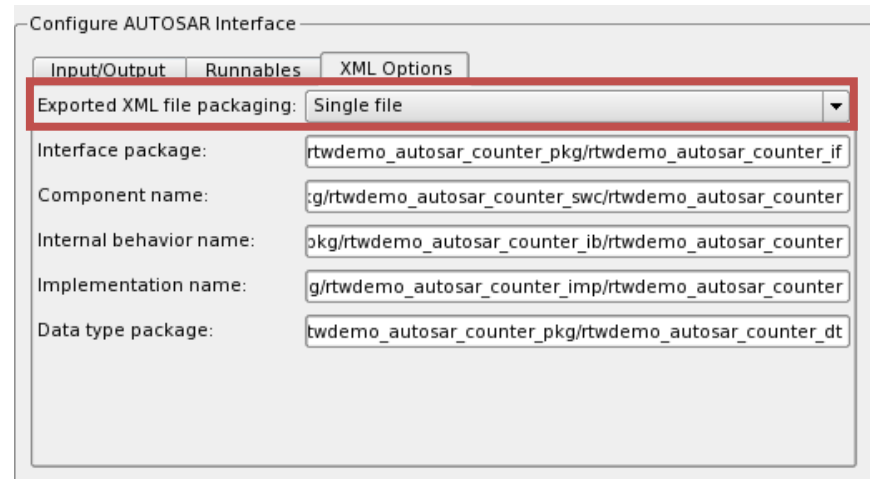
- Import, design, and export AUTOSAR Sensor/Actuator Components using Simulink
- Use a unified interface for application software and sensor/actuator components



# Export AUTOSAR XML as Single File

## Fewer files to maintain in version control

- New option to export XML as single file
- Existing option to export XML as modular (multiple) files



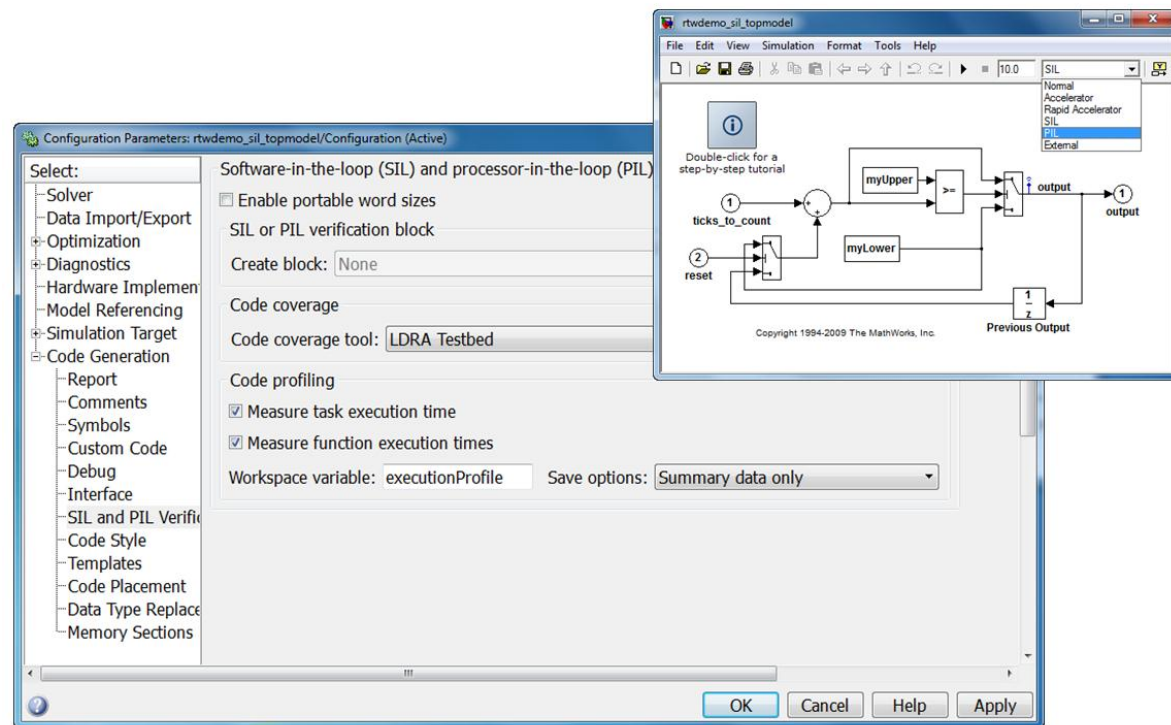
*modelName.arxml*



# LDRA Testbed Integration

## Measure code coverage during SIL and PIL testing

- Analyze generated code coverage using LDRA Testbed
- Launch LDRA code coverage report directly from MATLAB
- Supports DO-178 and other high-integrity workflows

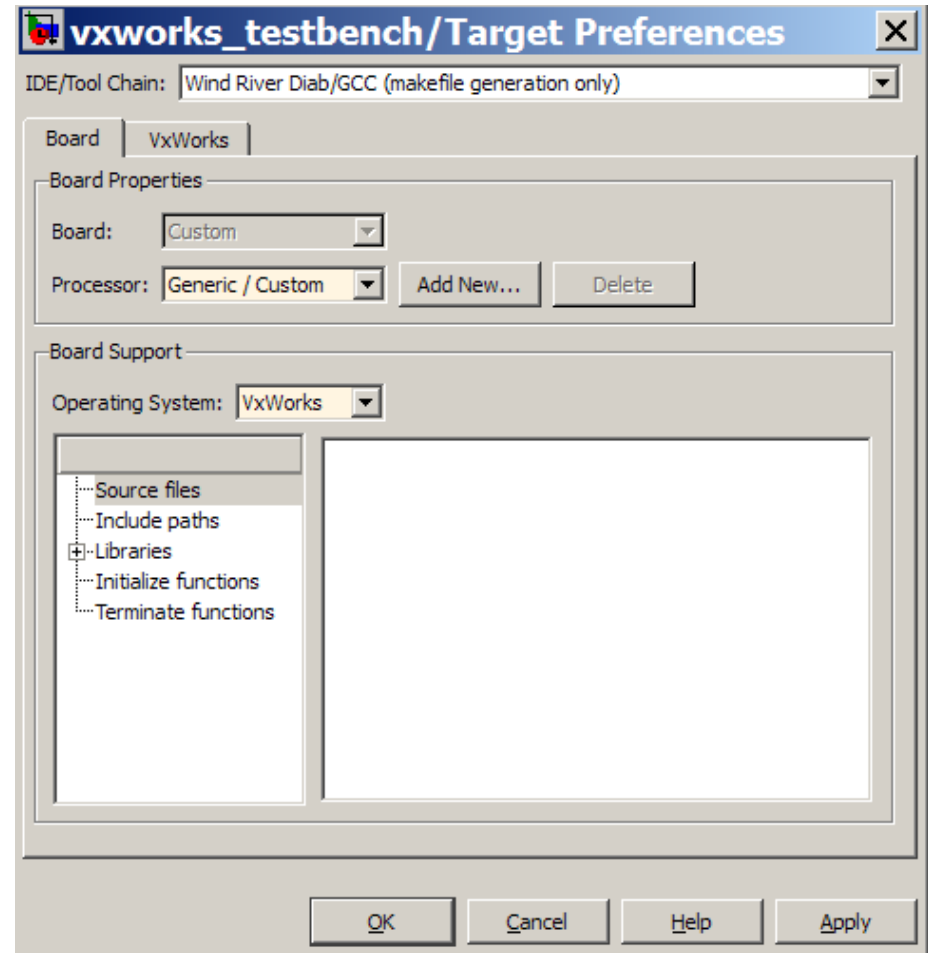


» rtwdemo\_sil\_topmodel

# Wind River VxWorks 6.8 Support Using Makefiles

## Use updated version of VxWorks Real-Time Operating System (RTOS)

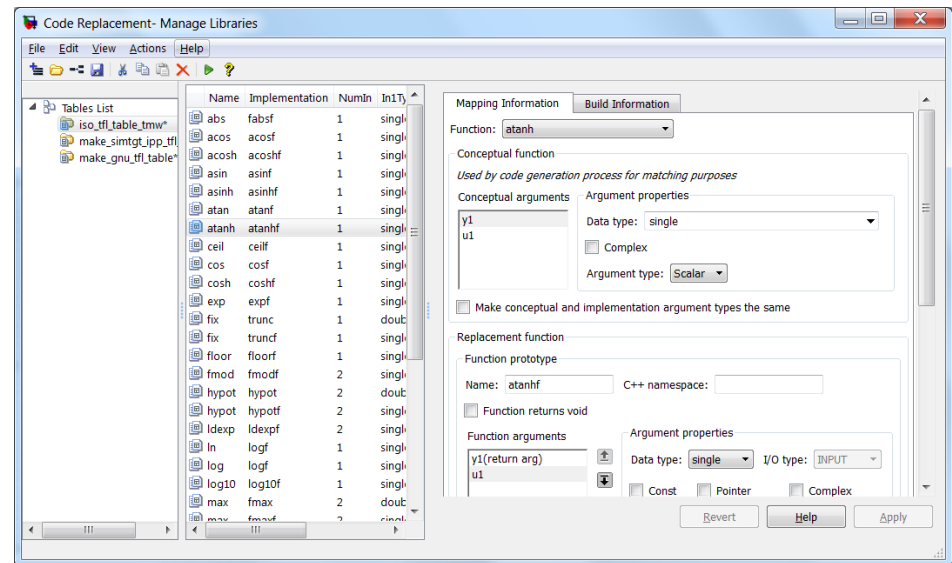
- Makefile support only
- Example integrations with other RTOS:
  - Embedded Linux
  - Green Hills INTEGRITY



# Code Replacement User Interface

## Design and manage code replacements using a graphical user interface

- Graphically create, import, modify, and validate code replacements
- Edit table entries more easily
- Register code replacements without writing MATLAB code

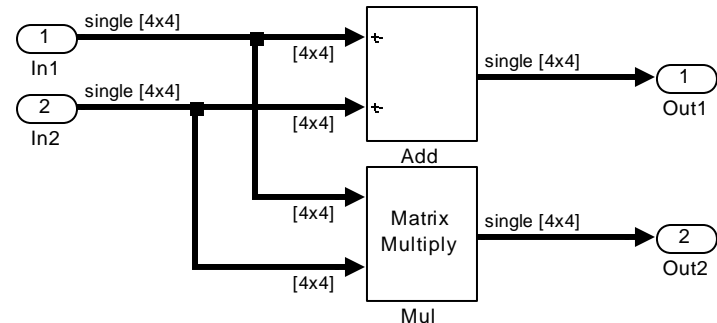


Note: The Code Replacement Library uses Target Function Library technology.

# Data Alignment in Code Replacements

## Boost code performance by leveraging SIMD and IPP

- Perform target optimizations that require data alignment
  - SIMD, Intel IPP, etc.
- Specify data alignment on a specific boundary using implementation function in code replacement library
- Use the implementation function to propagate the alignment to variable or structure type definition



```

void mDA_step(void)
{
    matrix_add_4x4s_w_da(mDA_U.In1, mDA_U.In2,
                        &mDA_B.Add[0]);
    .....
    matrix_mul_4x4s_w_da(mDA_U.In1, mDA_U.In2,
                        *(real32_T (*)[16])&mDA_Y.Out2[0]);
}
    
```

In model.c

```

typedef struct {
    __declspec(align(16)) real32_T Add[16];
} BlockIO_mDA;

typedef struct {
    __declspec(align(16)) real32_T In1[16];
    __declspec(align(16)) real32_T In2[16];
} ExternalInputs_mDA;

typedef struct {
    real32_T Out1[16];
    __declspec(align(16)) real32_T Out2[16];
} ExternalOutputs_mDA;
    
```

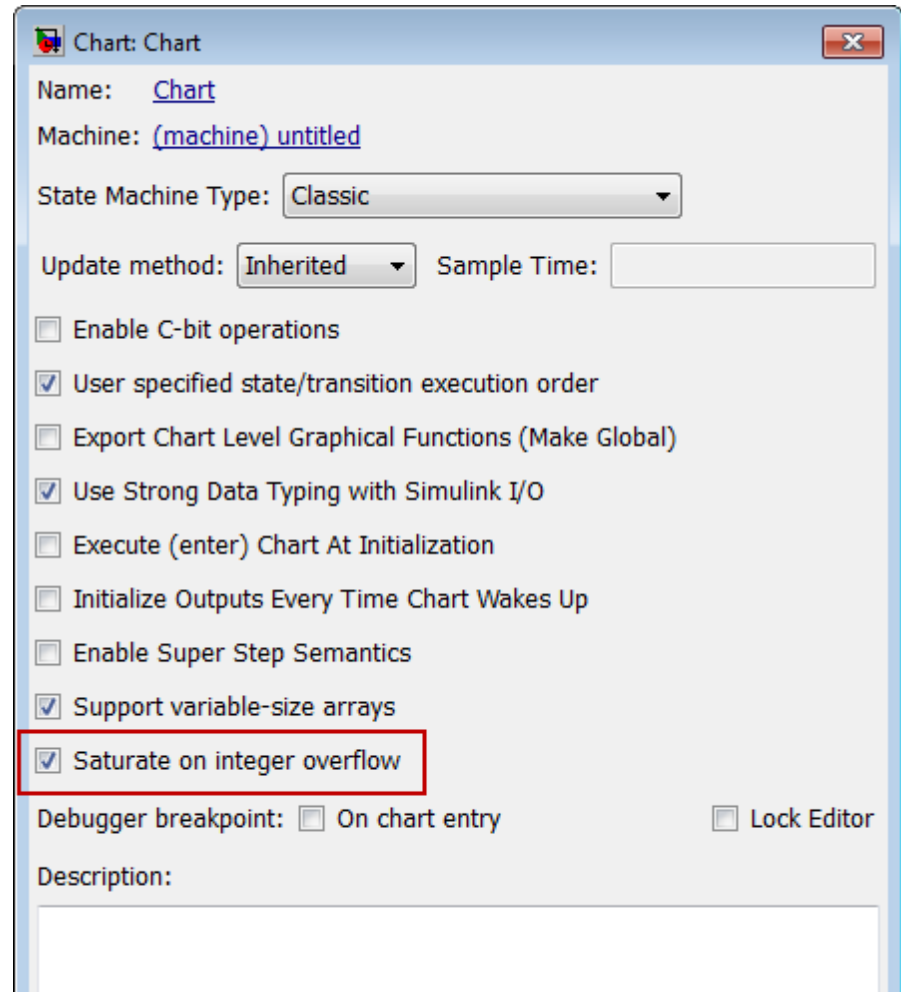
In model.h

» rtwdemo\_tflalign

# Saturation on Overflow in Stateflow

## Option to saturate on integer arithmetic overflow

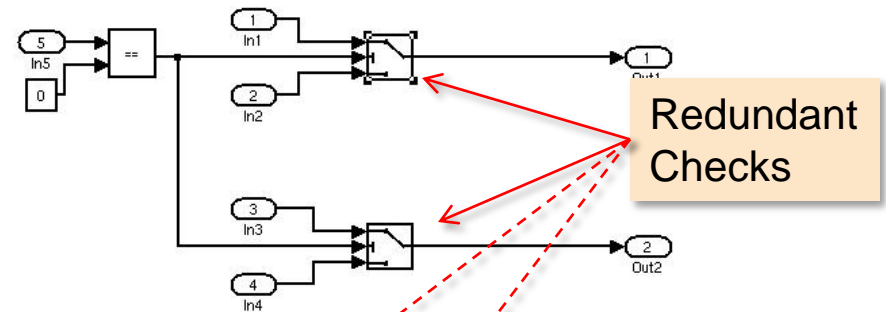
- Ensure consistency with MATLAB and Simulink integer arithmetic
- Enable saturation for greater robustness in safety-critical applications
- Disable saturation for faster execution



# Redundant Condition Check Elimination

Automatically remove redundant checks found in typical modeling patterns

- Reduces code size
- Improves execution speed
- Enables additional optimizations resulting from simplified control flow



```

if (x[0] > 0.0) {
    out1 = in1;
} else {
    out1 = in2;
}

```

```

if (x[0] > 0.0) {
    out2 = in3;
} else {
    out2 = in4;
}

```

R2011a

```

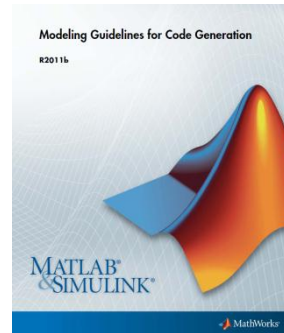
if (x[0] > 0.0) {
    out1 = in1;
    out2 = in3;
} else {
    out1 = in2;
    out2 = in4;
}

```

R2011b

# New Modeling Guidelines for C Code Generation

Leverage MathWorks and industry expertise to optimize generated code



- New code generation guidelines address modeling of shared memory
- Guidelines provide:
  - Concise actionable summary of recommendations
  - Rationale behind recommendations

## cgsl\_0104: Modeling global shared memory using data stores

ID: Title	cgsl_0104: Modeling global shared memory using data stores	
Description	When using data store blocks to model shared memory across multiple models:	
	A	In the Configuration Parameters dialog box, on the Diagnostics pane, set Data Validity > Data Store Memory Block > Duplicate data store names to error for all the models in the hierarchy
	B	Define the data store using a Simulink Signal or MPT Signal object
	C	Do not use Data Store Memory blocks in any of the models
Notes	<p>If multiple Data Store blocks use the same data store name within a model, then Simulink interprets each instance of the data store as having a unique local scope.</p> <p>Use the diagnostic Duplicate data store names to help detect unintended identifier reuse. For models intentionally using local data stores, set the diagnostic to warning. Verify that only intentional data stores are included.</p> <p>Merge blocks, used in conjunction with subsystems operating in a mutually exclusive manner, provide a second method of modeling global data across multiple models.</p>	
Rationale	A, B, C	Promotes a modeling pattern where a single consistent data store is used across all models and a single global instance is created in the generated code.

## cgsl\_0105: Modeling local shared memory using data stores

ID: Title	cgsl_0105: Modeling local shared memory using data stores	
Description	When using data store blocks as local shared memory:	
	A	Explicitly create the data store using a Data Store Memory block.
	B	Deselect the block parameter option Data store name must resolve to Simulink signal object.
	C	Consider following a naming convention for local Data Store Memory blocks.
Notes	<p>Use the diagnostic Duplicate data store names to help detect unintended identifier reuse. For models intentionally using local data stores, set the diagnostic to warning. Verify that only intentional data stores are included.</p> <p>Data store blocks are realized as global memory in the generated code. If they are not assigned a specific storage class, they are included in the DWork structure. In the model, the data store is scoped to the defining subsystem and below. In the generated code, the data store has file scope.</p>	
Rationale	A, B	Data store block is treated as a local instance of the data store
	C	Provides graphical feedback that the data store is local