

# A Unified Approach to Model and Code Verification

12 May 2016

**Chuck Olosky**  
**Anthony Abrham**

**Application Engineering**  
**Application Engineering**

# Motivation

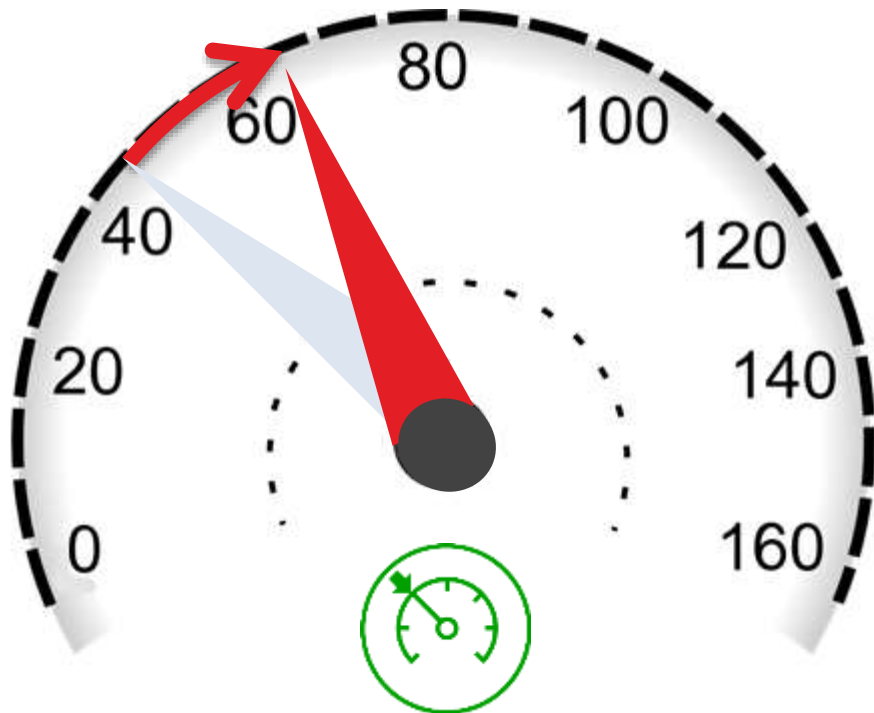
- Most controls applications are a combination of model-based generated code and hand code
- How do I efficiently test this mix of hand code and generated code?
- MathWorks has tools for testing models and tools for testing code
- Is there a workflow for me to use these tools in a complementary, optimum way?

# Agenda

- Static analysis of the model and code before functional testing
- Dynamic, functional testing of the model, s-function and generated code
- Static analysis of the integrated code:  
hand code, s-function code and generated code
- A unified, complementary model and code verification workflow to continually increase design confidence

# Case Study: Cruise Control Application

***Objective: set cruise control target speed and pedal position based on driver & vehicle inputs***



**65 mph**

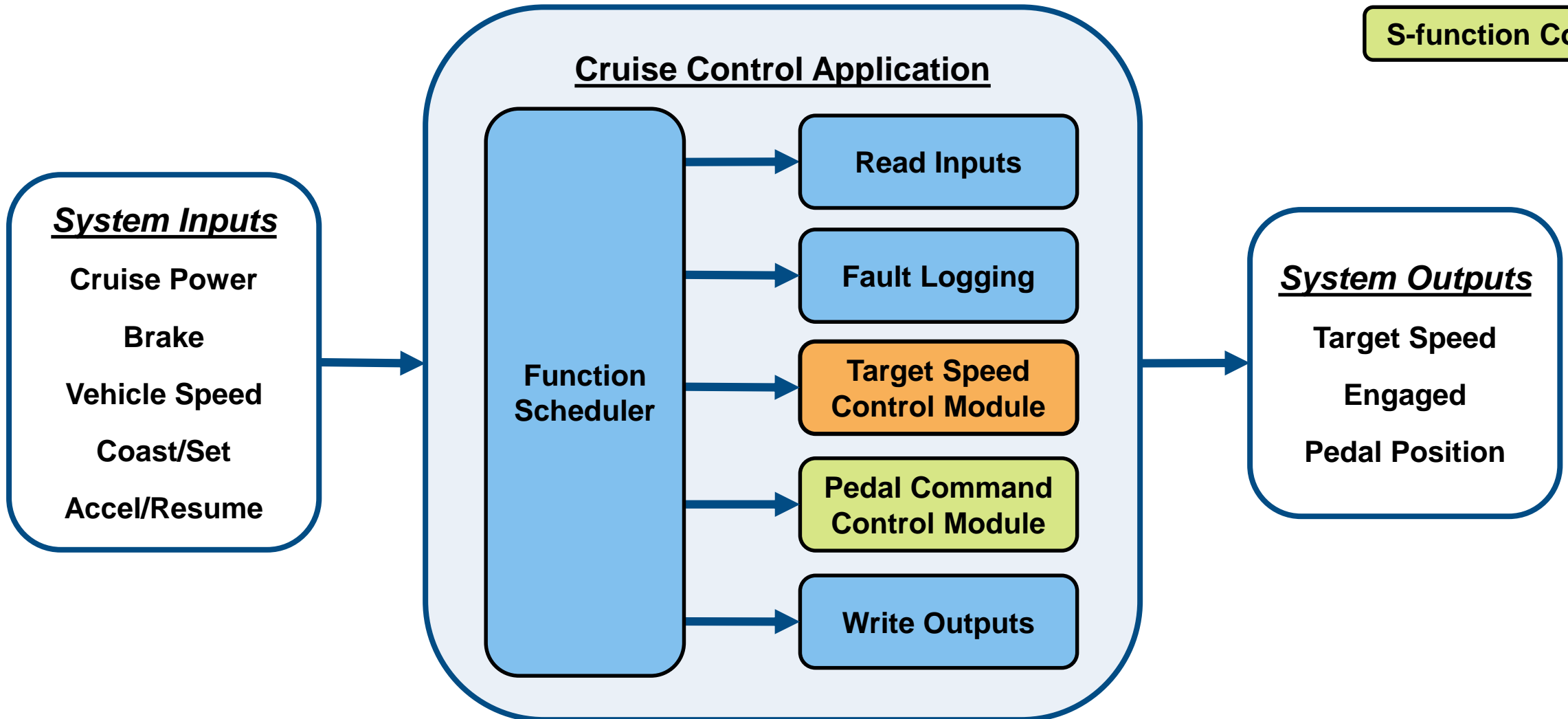
Cruise Control Application (C code)

- *Hand code components*
- *Model-based Stateflow component*
- *Model-based S-function component*



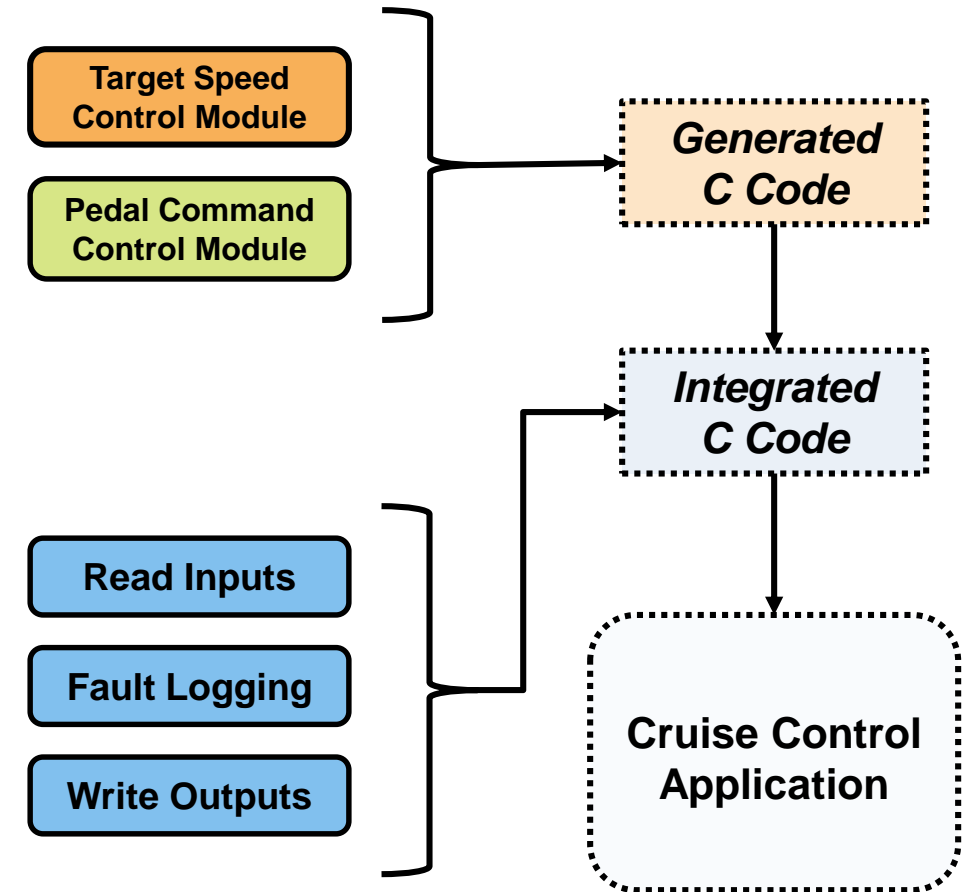
# Case Study: Cruise Control Architecture

- Hand Code
- MBD Gen Code
- S-function Code



# Case Study: Roles & Workflow

- *MBD Controls Guy: Chuck*
  - *Develops modules using Simulink models*
  - *Integrates C code with models via s-functions*
  - *Generates the code*
  - *Relies on model-based testing methods*
  
- *Integration & Build Guy: Anthony*
  - *Develop C code modules by hand*
  - *Integrates hand code and generated code*
  - *Creates the ECU build*
  - *Relies on the HiL bench for testing*



# Case Study: Deliver First Production Release to Customer

*To deliver our first production release we will need the following new features/changes:*

- **Move signals/cals from floats to integers in Target Speed Module**
- **Include customer lookup table code in Pedal Command to support calibration**
- **Demonstrate generated code is MISRA compliant**
- **Remove unused fault record**
- **Migrate the code to run on customer's ECU (14-bit to 12-bit ADC)**

*In addition to the changes we will need to provide functional test results for the model-based modules and the integrated code.*

## Model-based Design Tasks

*First let's focus on the model-based design tasks and what checks are available:*

- **Convert signals/cals from floats to integers in Target Speed Module**
- **Include the customer lookup table in the Pedal Cmd to support calibration**
- **Demonstrate generated code is MISRA compliant**

*Our approach will be to do checks before functional testing, early in the development to minimize re-work.*



# Floats to Integers: Checking the Model for Design Errors

**Configuration Parameters: CruiseControl\_IntCalc/ModelReferencin...**

Select:

- Solver
- Data Import/Export
- Optimization
- Diagnostics
- Hardware Implementation
- Model Referencing
- Simulation Target
- Code Generation
- Design Verifier
  - Block Replacements
  - Parameters
  - Test Generation
  - Design Error Detection**
  - Property Proving
  - Results
  - Report

Design Error Detection

- Dead logic
- Identify active logic
- Integer overflow
- Division by zero
- Check specified inter...
- Out of bound array ac...

**Simulink Design Verifier ...**

[Close results](#)

**Design error detection completed normally.**

2/70 objectives are dead logic.  
68/70 objectives are active logic.

**Results:**

- [Generate detailed analysis report](#)
- [Open harness model](#)

**Stateflow (chart) CruiseControl\_IntCalc/Compute target speed \* - Simulink**

File Edit View Display Chart Simulation Analysis Code Tools Help

CruiseControl\_IntCalc Compute target speed

OFF  
en: engaged = false;  
tspeed = 0;

CRUISE

STANDBY  
en: engaged=false;

ON  
en: engaged = true;

Accl  
tspeed = tspeed + incoDec;

Steady  
tspeed = Speed;

Coast  
tspeed = tspeed - incoDec;

View 4 warnings 81% FixedStepDiscrete

Simulink Design Verifier identifies model design errors that may result in “dead logic” that would prevent successful functional testing

# Root Cause Analysis/Fix of Dead Logic

```

Command Window
debug>> incdec
incdec =
    1

debug>> holdrate
holdrate =
    5

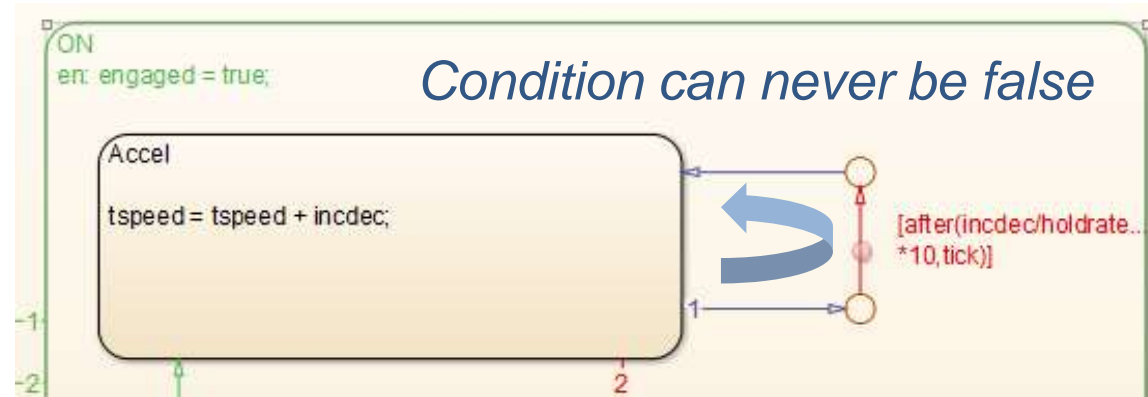
debug>> class(incdec)
ans =
    uint8

debug>> class(holdrate)
ans =
    uint8

debug>> incdec/holdrate*10
ans =
    0

debug>> 10*incdec/holdrate
ans =
    2

fx debug>>
    
```



- **Dead logic** due to “uint8” operation on **incdec/holdrate\*10**
- **Fix** change the order of operation **10\*incdec/holdrate**

## Model-based Design Tasks

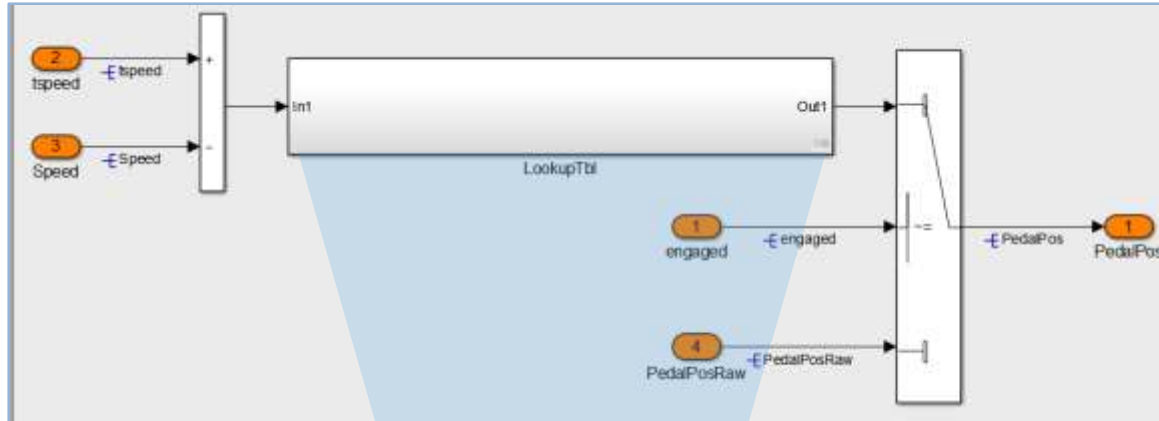
*First let's focus on the model-based design tasks and what checks are available:*

- **Convert signals/cals from floats to integers in Target Speed Module**
- **Include the customer lookup table in the Pedal Cmd to support calibration**
- **Demonstrate generated code is MISRA compliant**

*Our approach will be to do checks before functional testing, early in the development to minimize re-work*

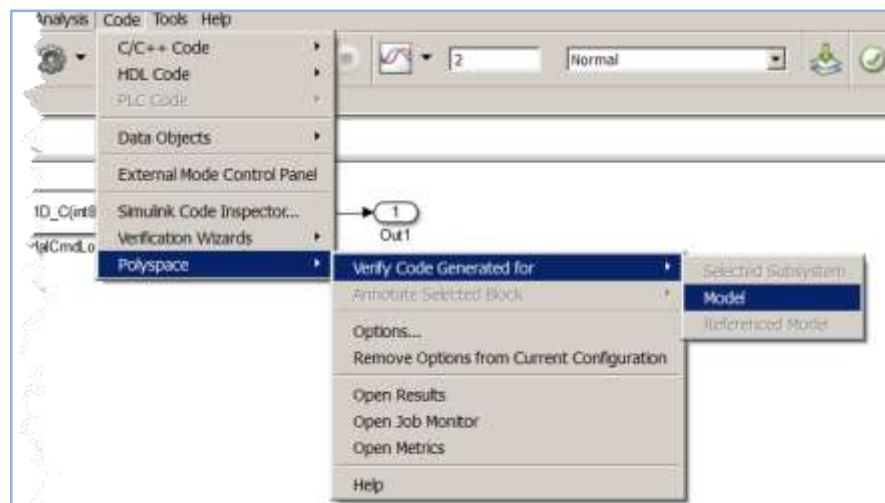
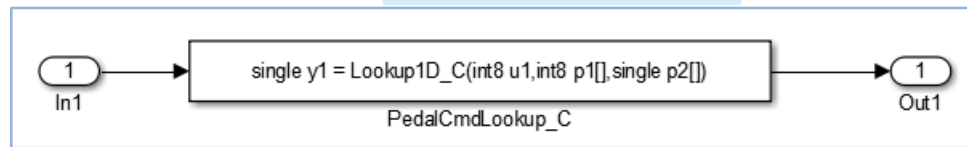
# Customer Lookup Table: Checking the S-Function Code for Runtime Errors

Pedal Command Control Module



```

Source
Lookup1D_C.c x
23     y = Y[0];
24     }
25     else
26     {
27     while((u >= X[index]) && (index < (mySize)))
28     {
29         index++;
    
```



Result Details

Variable trace

Result Review

Severity  *Enter comment here...*

Status

**! Pointer access out of bounds** (Impact: High) ?  
Attempt to dereference pointer at index 11.  
Valid range: [0 .. 10]

	Event	File	Scope	Line
1	! Pointer access out of bounds	Lookup1D_C.c	Lookup1D_C()	27

# Root Cause Analysis/Fix of S-Function Run-time Errors

```

25  /* Definition for custom storage class: Global */
26  real32_T PedalCmdY[11] = { 0.0F, 0.5F, 1.0F, 1.5F, 2.0F, 2.5F,
27                            3.0F, 3.5F, 4.0F, 4.5F, 5.0F };
28  int8_T SpeedDelX[11] = { -5, -4, -3, -2, -1, 0, 1, 2, 3, 4, 5 };
29
54  * S-Function (PedalCmdLookup)
55  * Sum: '<Root>/Sum'
56  */
57  if (engaged) {
58      PedalPos = Lookup1D_C( (int8_T)rtb_Sum, (int8_T*)(&(SpeedDelX[0])),
59                          (real32_T*)(&(PedalCmdY[0])));
60  } else {
61      PedalPos = PedalPosRaw;
62  }
  
```

```

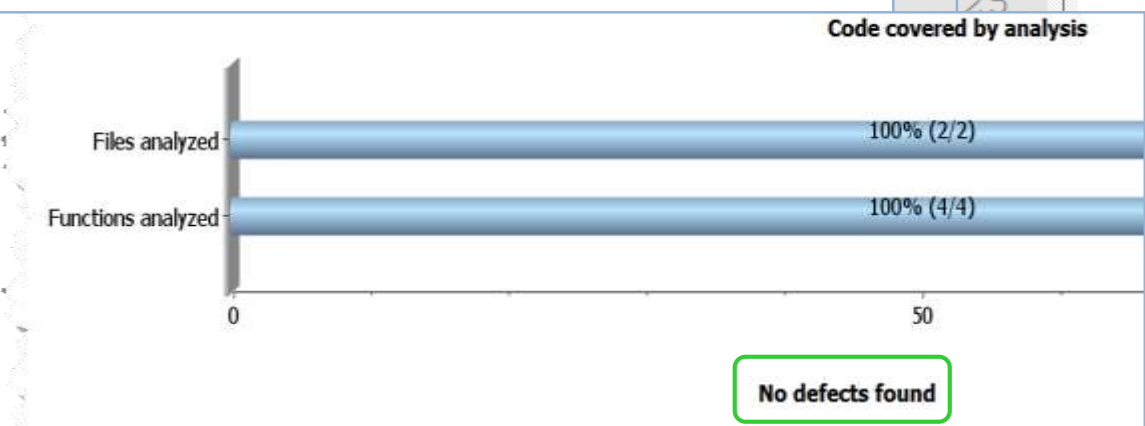
9  float Lookup1D_C(char u, char const X[], float const Y[])
10  {
11      float y = 0.0f;
12      unsigned char index = 0;
13      float temp = 0.0f;
14
15      unsigned char mySize = 11;
  
```

```

26      while((u >= X[index]) && (index < mySize))
27      {
28          index++;
  
```

```

25      else
26      {
27          while((u >= X[index]) && (index < (mySize-1)))
28          {
29              index++;
30          }
31          if (index > 0)
  
```



## Model-based Design Tasks

*First let's focus on the model-based design tasks and what checks are available:*

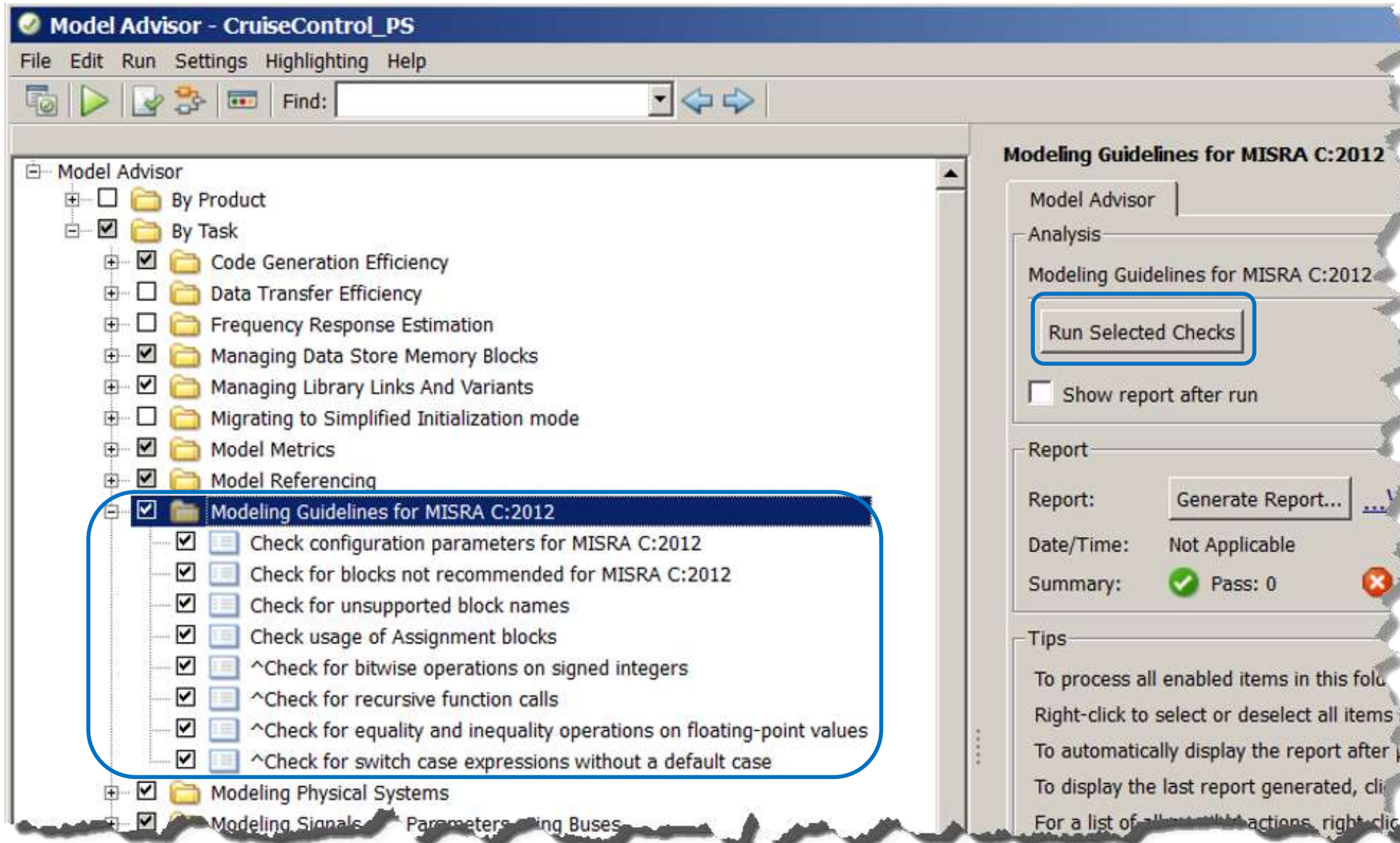
- **Convert signals/cals from floats to integers in Target Speed Module**
- **Include the customer lookup table in the Pedal Cmd to support calibration**
- **Demonstrate generated code is MISRA compliant**

*Our approach will be to do checks before functional testing, early in the development to minimize re-work*



# Checking Model for MISRA compliance with Model Advisor

Target Speed  
Control Module



The screenshot shows the Model Advisor application window titled "Model Advisor - CruiseControl\_PS". The interface includes a menu bar (File, Edit, Run, Settings, Highlighting, Help) and a toolbar with icons for running, saving, and finding. The main area is divided into a tree view on the left and a right-hand panel.

**Model Advisor Tree View:**

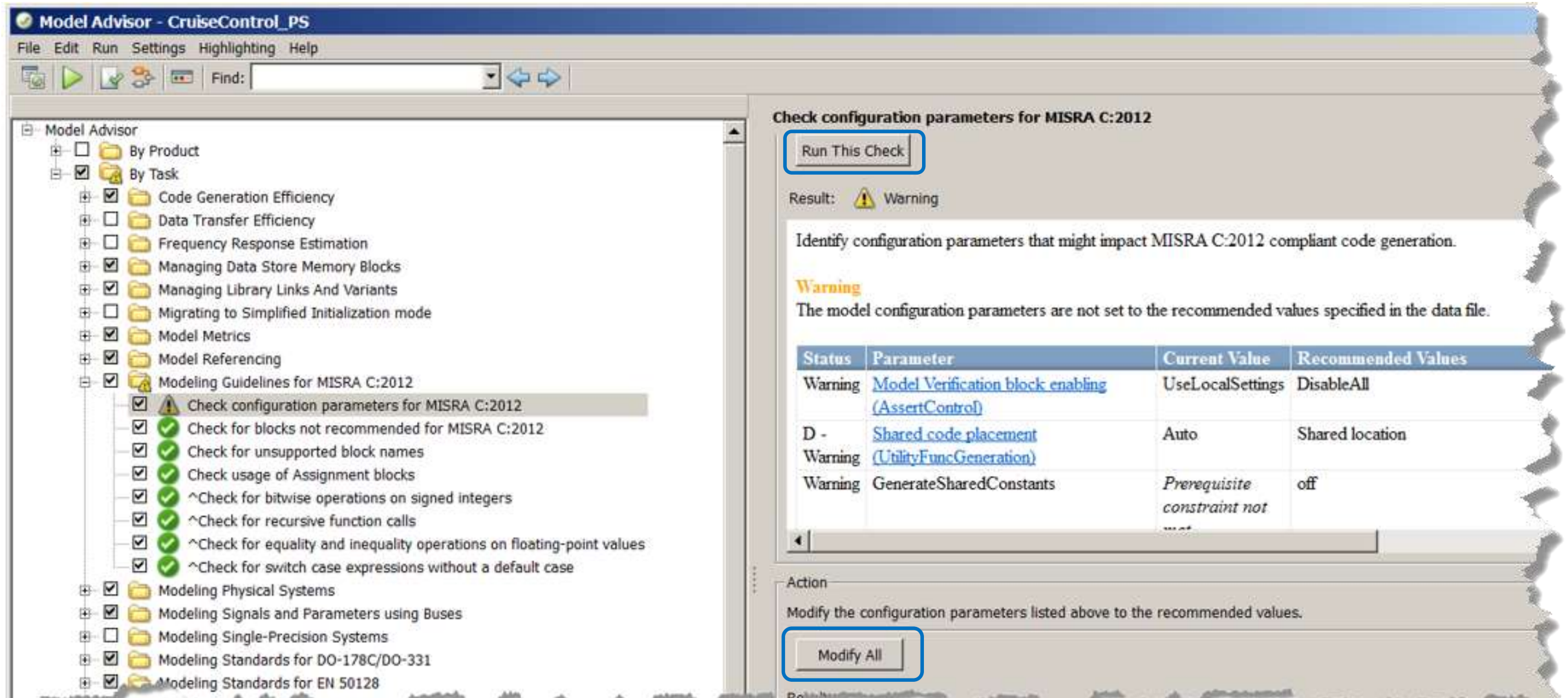
- Model Advisor
  - By Product
  - By Task
    - Code Generation Efficiency
    - Data Transfer Efficiency
    - Frequency Response Estimation
    - Managing Data Store Memory Blocks
    - Managing Library Links And Variants
    - Migrating to Simplified Initialization mode
    - Model Metrics
    - Model Referencing
    - Modeling Guidelines for MISRA C:2012** (highlighted with a blue circle)
      - Check configuration parameters for MISRA C:2012
      - Check for blocks not recommended for MISRA C:2012
      - Check for unsupported block names
      - Check usage of Assignment blocks
      - ^Check for bitwise operations on signed integers
      - ^Check for recursive function calls
      - ^Check for equality and inequality operations on floating-point values
      - ^Check for switch case expressions without a default case
    - Modeling Physical Systems
    - Modeling Signals and Parameters on Buses

**Modeling Guidelines for MISRA C:2012 Panel:**

- Model Advisor
- Analysis
  - Modeling Guidelines for MISRA C:2012
    - Run Selected Checks** (button highlighted with a blue circle)
    - Show report after run
  - Report
    - Report:
    - Date/Time: Not Applicable
    - Summary: ✔ Pass: 0 ✘
  - Tips
    - To process all enabled items in this folder, click the Run Selected Checks button.
    - Right-click to select or deselect all items.
    - To automatically display the report after the analysis, check the Show report after run checkbox.
    - To display the last report generated, click the Generate Report button.
    - For a list of all available actions, right-click on any item in the tree view.

# Checking Model for MISRA compliance with Model Advisor

Target Speed  
Control Module



**Model Advisor - CruiseControl\_PS**


File Edit Run Settings Highlighting Help

Find: [ ]

**Model Advisor**

- By Product
- By Task
  - Code Generation Efficiency
  - Data Transfer Efficiency
  - Frequency Response Estimation
  - Managing Data Store Memory Blocks
  - Managing Library Links And Variants
  - Migrating to Simplified Initialization mode
  - Model Metrics
  - Model Referencing
  - Modeling Guidelines for MISRA C:2012
    - Check configuration parameters for MISRA C:2012**
    - Check for blocks not recommended for MISRA C:2012
    - Check for unsupported block names
    - Check usage of Assignment blocks
    - ^Check for bitwise operations on signed integers
    - ^Check for recursive function calls
    - ^Check for equality and inequality operations on floating-point values
    - ^Check for switch case expressions without a default case
  - Modeling Physical Systems
  - Modeling Signals and Parameters using Buses
  - Modeling Single-Precision Systems
  - Modeling Standards for DO-178C/DO-331
  - Modeling Standards for EN 50128

**Check configuration parameters for MISRA C:2012**

Result:  Warning

Identify configuration parameters that might impact MISRA C:2012 compliant code generation.

**Warning**

The model configuration parameters are not set to the recommended values specified in the data file.

Status	Parameter	Current Value	Recommended Values
Warning	<a href="#">Model Verification block enabling (AssertControl)</a>	UseLocalSettings	DisableAll
D -	<a href="#">Shared code placement (UtilityFuncGeneration)</a>	Auto	Shared location
Warning	GenerateSharedConstants	Prerequisite constraint not met	off

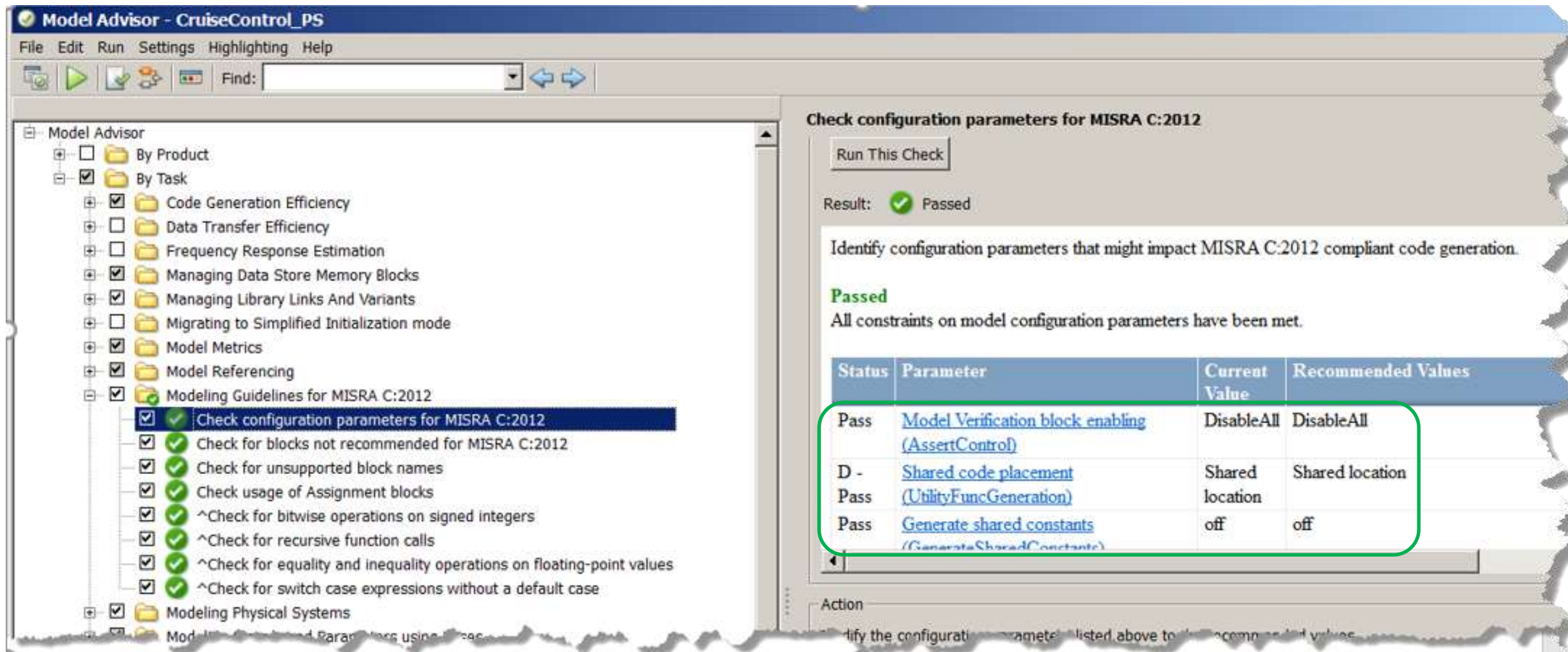
Action

Modify the configuration parameters listed above to the recommended values.



# Checking Model for MISRA compliance with Model Advisor

Target Speed Control Module



**Model Advisor - CruiseControl\_PS**

File Edit Run Settings Highlighting Help

Find: [ ]

**Model Advisor**

- By Product
- By Task
  - Code Generation Efficiency
  - Data Transfer Efficiency
  - Frequency Response Estimation
  - Managing Data Store Memory Blocks
  - Managing Library Links And Variants
  - Migrating to Simplified Initialization mode
  - Model Metrics
  - Model Referencing
  - Modeling Guidelines for MISRA C:2012
    - Check configuration parameters for MISRA C:2012**
    - Check for blocks not recommended for MISRA C:2012
    - Check for unsupported block names
    - Check usage of Assignment blocks
    - ^Check for bitwise operations on signed integers
    - ^Check for recursive function calls
    - ^Check for equality and inequality operations on floating-point values
    - ^Check for switch case expressions without a default case
  - Modeling Physical Systems
  - Modeling Parameters using...

**Check configuration parameters for MISRA C:2012**

Run This Check

Result: ✔ Passed

Identify configuration parameters that might impact MISRA C:2012 compliant code generation.

**Passed**

All constraints on model configuration parameters have been met.

Status	Parameter	Current Value	Recommended Values
Pass	<a href="#">Model Verification block enabling (AssertControl)</a>	DisableAll	DisableAll
D - Pass	<a href="#">Shared code placement (UtilityFuncGeneration)</a>	Shared location	Shared location
Pass	<a href="#">Generate shared constants (GenerateSharedConstants)</a>	off	off

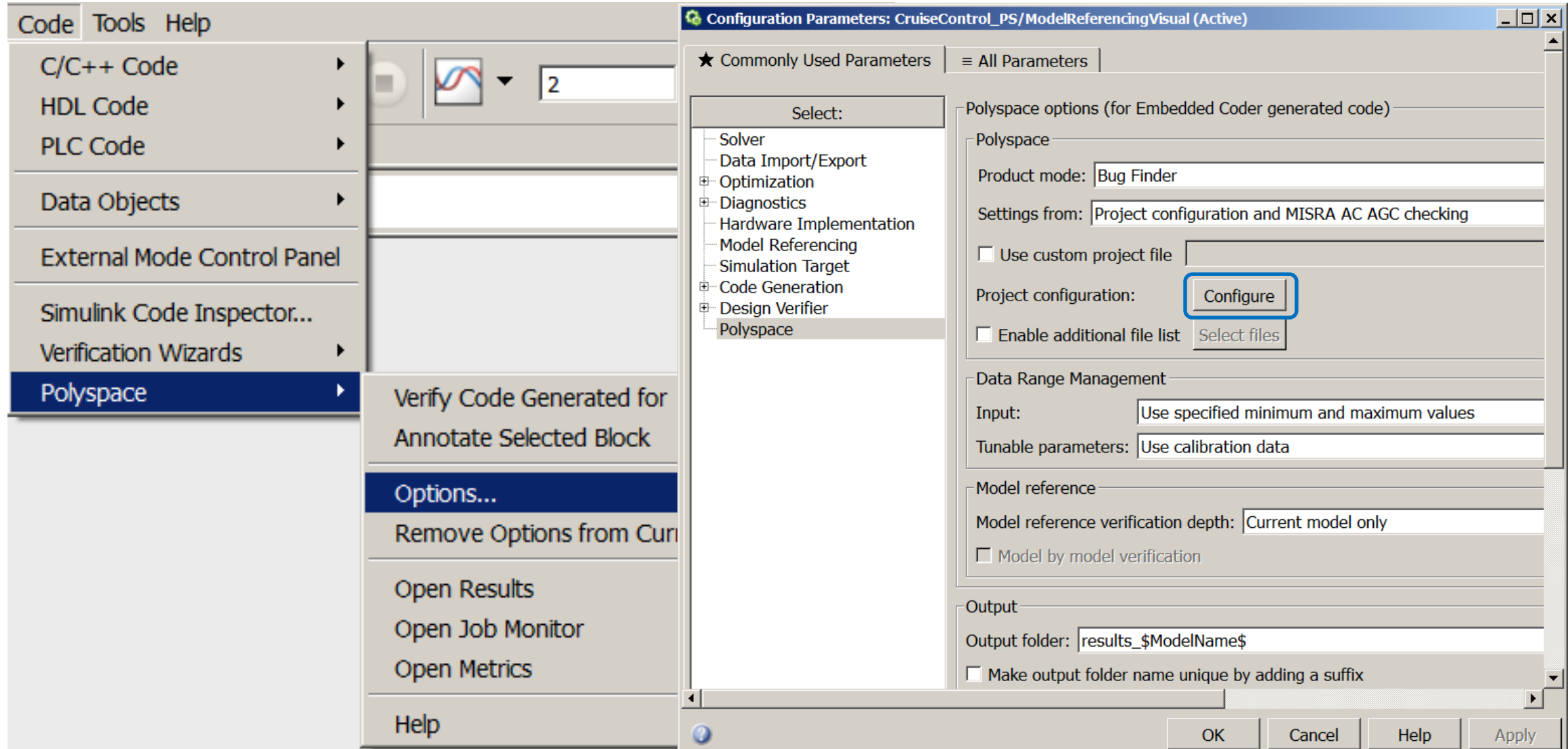
Action

Modify the configuration parameters listed above to the recommended values.

- ✔ Checks model design and code configuration settings
- ✔ Increases likelihood of generating MISRA C:2012 compliant code

# Configuring Polyspace from the Model

**Target Speed Control Module**



The screenshot shows the MATLAB interface with the Polyspace menu open and the Configuration Parameters dialog box for 'CruiseControl\_PS/ModelReferencingVisual' active.

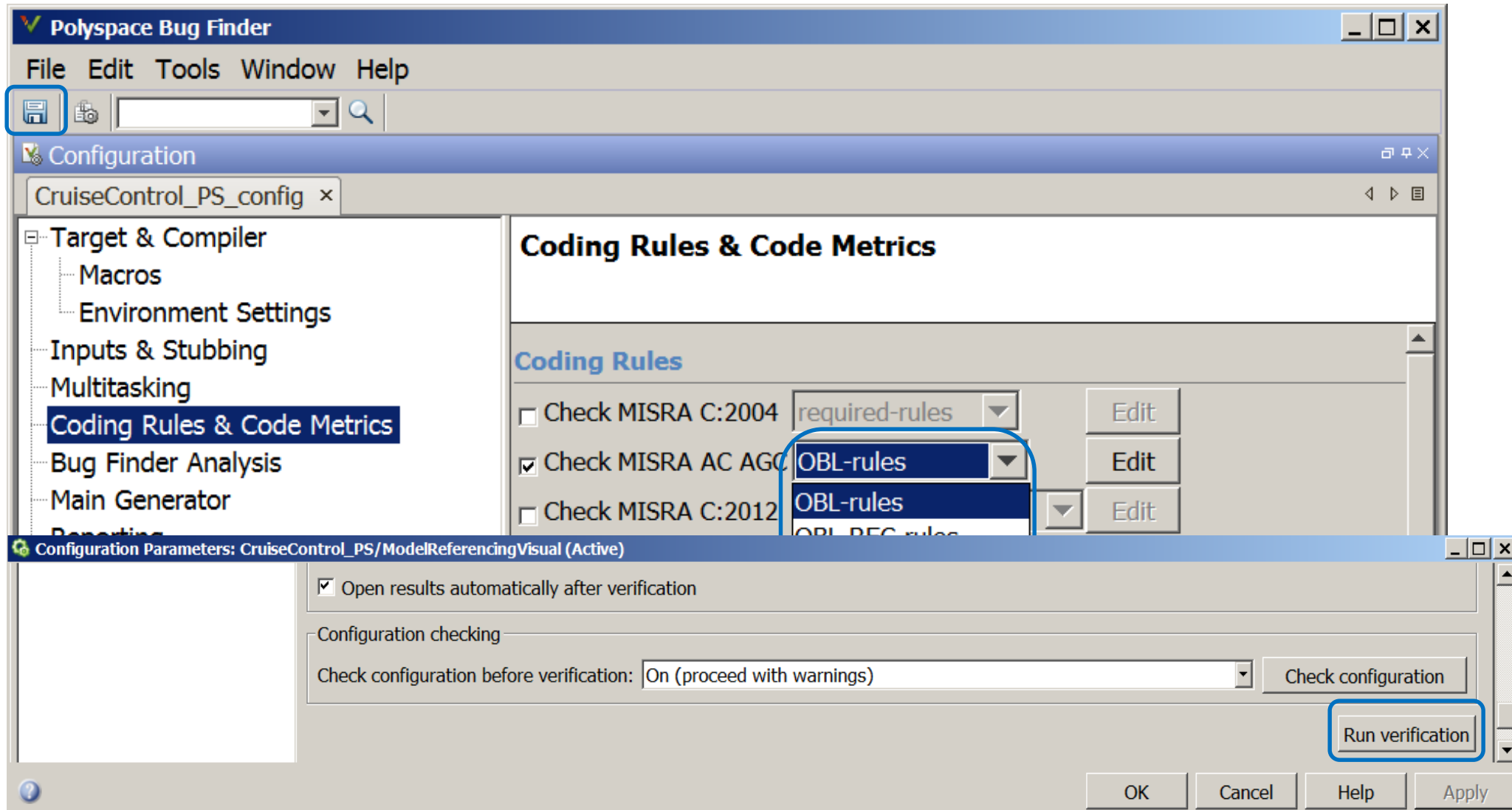
**Polyspace Menu:**

- Verify Code Generated for Annotate Selected Block
- Options...**
- Remove Options from Current Model
- Open Results
- Open Job Monitor
- Open Metrics
- Help

**Configuration Parameters Dialog:**

- Commonly Used Parameters:** Solver, Data Import/Export, Optimization, Diagnostics, Hardware Implementation, Model Referencing, Simulation Target, Code Generation, Design Verifier, Polyspace.
- Polyspace options (for Embedded Coder generated code):**
  - Product mode: Bug Finder
  - Settings from: Project configuration and MISRA AC AGC checking
  - Use custom project file
  - Project configuration: **Configure**
  - Enable additional file list **Select files**
  - Data Range Management:
    - Input: Use specified minimum and maximum values
    - Tunable parameters: Use calibration data
  - Model reference:
    - Model reference verification depth: Current model only
    - Model by model verification
  - Output:
    - Output folder: results\_ \$ModelName\$
    - Make output folder name unique by adding a suffix

# Launching Polyspace from the Model



The screenshot shows the Polyspace Bug Finder Configuration dialog box. The window title is "Polyspace Bug Finder". The menu bar includes "File", "Edit", "Tools", "Window", and "Help". The "Configuration" pane is open, showing a tree view on the left with "Coding Rules & Code Metrics" selected. The main area displays "Coding Rules & Code Metrics" with a list of rules:

- Check MISRA C:2004 required-rules Edit
- Check MISRA AC AGC OBL-rules Edit
- Check MISRA C:2012 OBL-rules Edit
- OBL-DEC rules Edit

The "Check MISRA AC AGC" rule is checked, and its dropdown menu is open, showing "OBL-rules" selected. The "Run verification" button is highlighted with a red box. Other buttons include "OK", "Cancel", "Help", and "Apply".



# Review Bug Finder MISRA results

Target Speed  
Control Module

The screenshot displays the Polyspace Bug Finder interface. The main window shows a tree view of results under 'MISRA AGC 18', with '8 Declarations and definitions 14' highlighted. The 'Result Details' pane shows a selected MISRA rule: 'MISRA AC AGC 8.10 (Obligatory)'. The description of the rule is: 'All declarations and definitions of objects or functions at file scope shall have internal linkage unless external linkage is required. Variable 'AccelResSw' should have internal linkage.' The 'Source' pane shows the corresponding C code snippet:

```

38 /* Definition for custom storage class: Global */
39 boolean_T AccelResSw;
40 boolean_T Brake;
41 boolean_T CoastSetSw;
42 boolean_T CruiseOnOff;
43 uint8_T Speed;

```

# Reduce MISRA violations with “Code Placement” setting

Target Speed  
Control Module

★ Commonly Used Parameters | ≡ All Parameters

Select:

- Solver
- Data Import/Export
- ⊕ Optimization
- ⊕ Diagnostics
- Hardware Implementation
- Model Referencing
- Simulation Target
- ⊖ Code Generation
  - Report
  - Comments
  - Symbols
  - Custom Code
  - Interface
  - Code Style
  - Verification
  - Templates
  - Code Placement**
  - Data Type Replacement
  - Memory Sections

Global data placement (custom storage classes only)

Data definition:

Data definition filename:

Data declaration:

Data declaration filename:

#include file delimiter:

Use owner from data object for data definition placement

Global data placement (MPT data objects only)

Signal display level:

Code Packaging

File packaging format:

# Justify other violations by adding annotation

Target Speed  
Control Module

Polyspace Bug Finder - CruiseControl\_PS C:\Demo\VnV\VnV\_Wkshp\Work\results\_CruiseControl\_PS\CruiseControl\_PS

File Reporting Metrics Tools Window Help

Run Stop

Results Summary

All results \* New Showing 7/7

Family	ID	Information
MISRA AGC 7		
1 Environment 4		
8 Declarations and definitions 3		
8.10 All declarations and definitions of objects or functions		
5	Category: Obligatory	CruiseControl_PS.c
6	Category: Obligatory	CruiseControl_PS.c
7	Category: Obligatory	CruiseControl_PS.c

Result Details

Variable trace

Result Review

Severity: Not a defect Add justification here

Status: Justify with annotations

**ID 5: MISRA AC AGC 8.10** (Obligatory) ?

All declarations and definitions of objects or functions at file scope shall have internal linkage.

Configuration Result Details

Source

CruiseControl\_PS.c

```

29  /* Block states (auto storage) */
30  DW_CruiseControl_PS_T DW;
31
32  /* Real-time model */
33  RT_MODEL_CruiseControl_PS_T M;
34  RT_MODEL_CruiseControl_PS_T *const M = &M;

```

## Model-based Design Tasks

*First let's focus on the model-based design tasks and what checks are available:*

- **Convert signals/cals from floats to integers in Target Speed Module**
- **Include the customer lookup table in the Pedal Cmd to support calibration**
- **Demonstrate generated code is MISRA compliant**

*Our approach will be to do checks before functional testing, early in the development to minimize re-work*

## Model-based Design Tests

*All checks are complete, we will need to provide test results for the model-based modules:*

- **Functional testing of s-function based Pedal Command module**
- **Equivalence (model-to-code) testing of the Target Speed module**



# Functional Testing of Pedal Command (S-Function)

Pedal Command Control Module

Coverage analysis for the model and the s-function code.

```

25     {
26         while((u >= X[index]) && (index < (mySize-1)))
27         {
28             index++;

```

**Uncovered Links:** ➡

Metric	Coverage
Decision (D1)	100% (2/2) decision outcomes
Condition (C1)	75% (3/4) condition outcomes

**Conditions analyzed:**

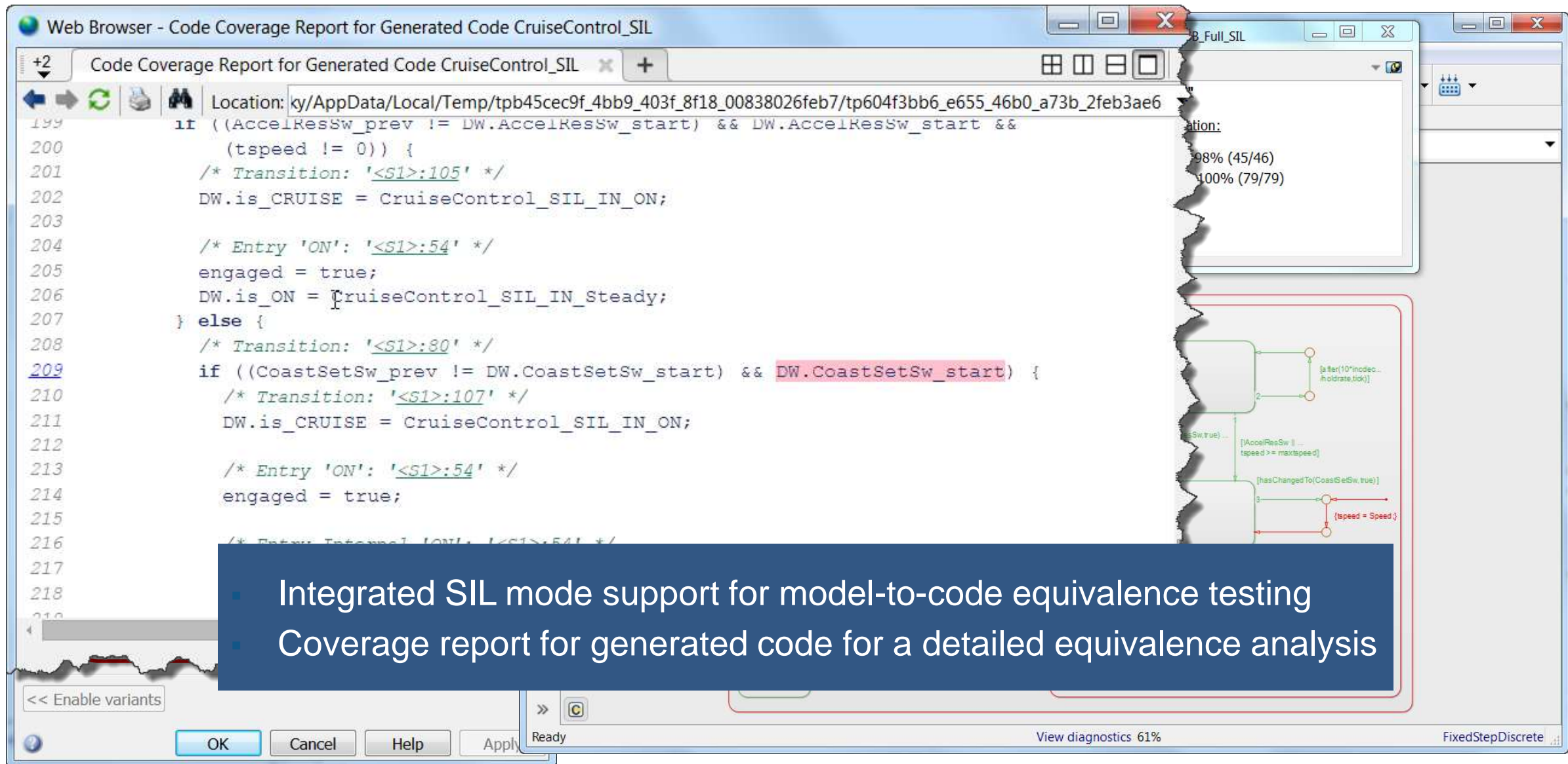
Description:	True	False
<code>u &gt;= X[index]</code>	105	21
<code>index &lt; (mySize-1)</code>	105	0

## Model-based Design Tests

*All checks are complete, we will need to provide test results for the model-based modules:*

- **Functional testing of s-function based Pedal Command module**
- **Equivalence (model-to-code) testing of the Target Speed module**

# Check the Generated Code for Equivalent Model Behavior



Web Browser - Code Coverage Report for Generated Code CruiseControl\_SIL

Code Coverage Report for Generated Code CruiseControl\_SIL

Location: ky/AppData/Local/Temp/tpb45cec9f\_4bb9\_403f\_8f18\_00838026feb7/tp604f3bb6\_e655\_46b0\_a73b\_2feb3ae6

```

199  if ((AccelResSw_prev != DW.AccelResSw_start) && DW.AccelResSw_start &&
200      (tspeed != 0)) {
201      /* Transition: '<S1>:105' */
202      DW.is_CRUISE = CruiseControl_SIL_IN_ON;
203
204      /* Entry 'ON': '<S1>:54' */
205      engaged = true;
206      DW.is_ON = CruiseControl_SIL_IN_Steady;
207  } else {
208      /* Transition: '<S1>:80' */
209      if ((CoastSetSw_prev != DW.CoastSetSw_start) && DW.CoastSetSw_start) {
210          /* Transition: '<S1>:107' */
211          DW.is_CRUISE = CruiseControl_SIL_IN_ON;
212
213          /* Entry 'ON': '<S1>:54' */
214          engaged = true;
215
216          /* Entry 'Internal ON': '<S1>:54' */
217
218
219

```

98% (45/46)  
100% (79/79)

View diagnostics 61%

FixedStepDiscrete

- Integrated SIL mode support for model-to-code equivalence testing
- Coverage report for generated code for a detailed equivalence analysis

## Model-based Design Tests

*All checks are complete, we will need to provide test results for the model-based modules:*

- **Functional testing of s-function based Pedal Command module**
- **Equivalence (model-to-code) testing of the Target Speed module**

# Integrated Code Testing

*The hand code design tasks:*

- **Remove unused fault record**
- **Migrate the code run on customer's ECU (14-bit to 12-bit ADC)**

*The minor hand code changes have been made.*

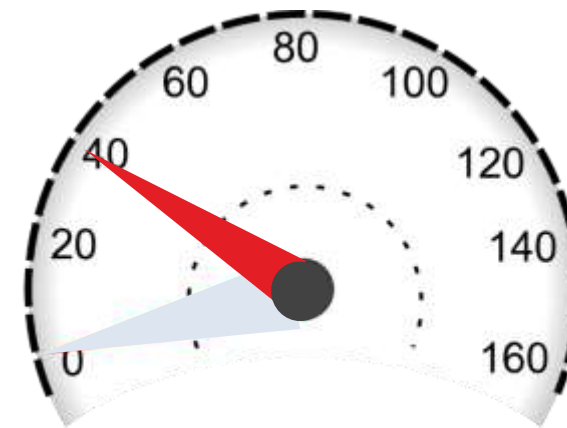
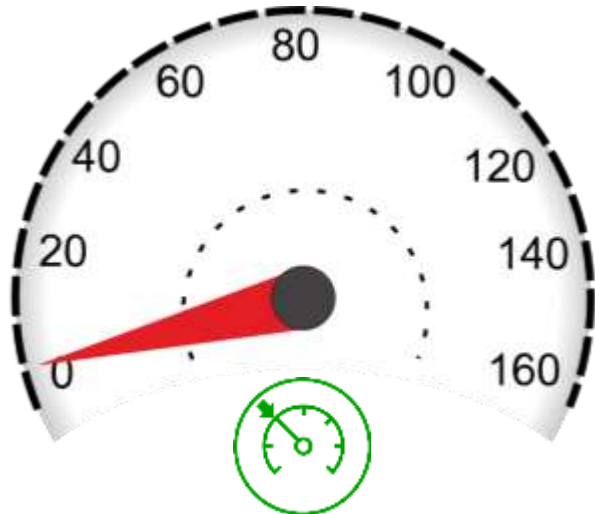
*An ECU build was created based on the integration of hand code and generated code*

**We now need to provide functional test results for the integrated code on the HiL bench**

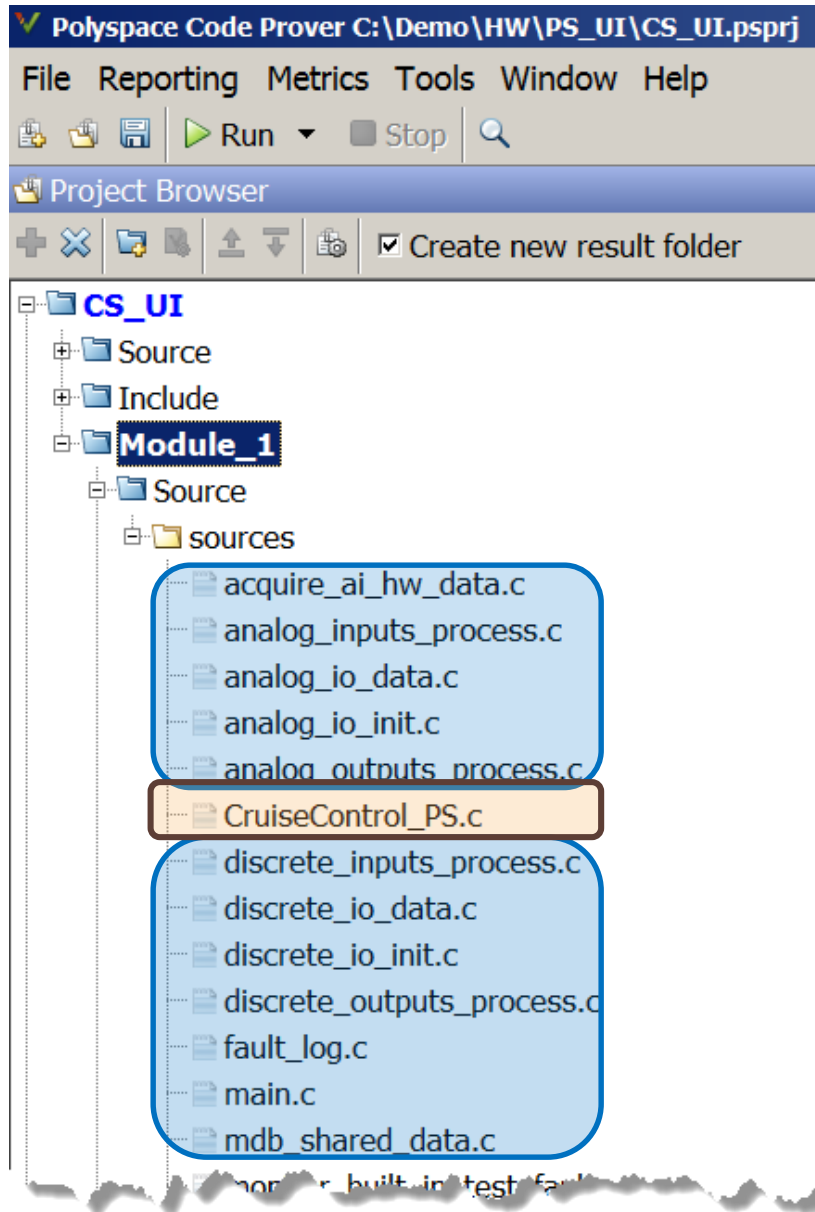
Find issues that result from the integration of tested modules from hand code, s-function code and model-based generated code.

## Issues Found on HIL Bench...

- *The Cruise Control powered off during fault testing*
- *And, the Target Speed never exceeded 40 mph*



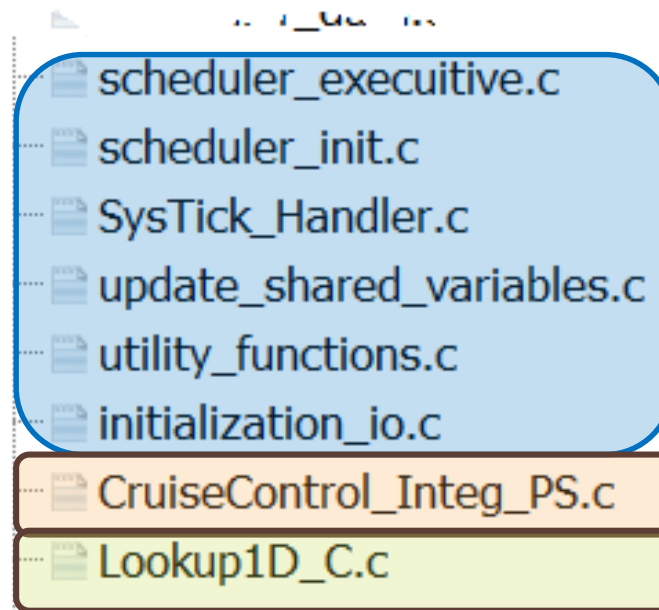
# Creating a Code Prover project to check the Integrated Code



- Read Inputs
- Write Outputs
- Scheduler
- Fault Logging

Target Speed Control Module

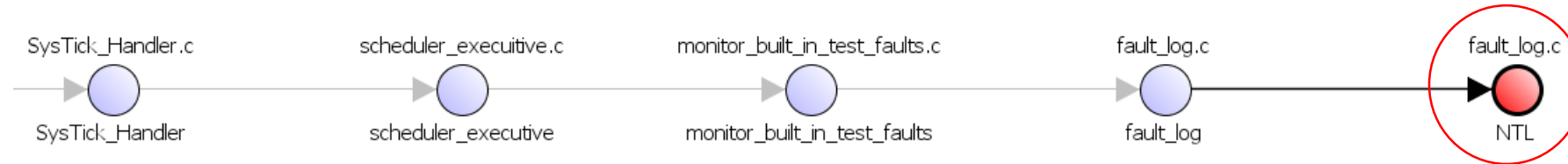
Pedal Command Control Module





# Code Integration Check with Polyspace: Non-terminating loop in Hand Code

Fault Logging



```

18 void fault_log(FAULT_LOG_INFO_T *pFaultInfo)
19 {
20     uint32_t ix;
21
22     uint32_t *pFlt = (uint32_t *)pFaultInfo;
23
24     /* Validate current fault index counter */
25     if (FaultRecIndexCntr >= (MAX_FAULT_LOG_INFO_SIZE - 12u))
26     {
27         FaultRecIndexCntr = 0x0u;
28     }
29
30     /* Store the current fault into circular buffer */
31     for (ix = 0; ix < 12u; ix++)
32     {
33         FaultInfoElement[FaultRecIndexCntr + ix] = *pFlt;
34         *pFlt = 0x0u;
35         pFlt++;
36     }
37     /* Update the circular buffer fault index counter */
38     FaultRecIndexCntr += ix-1;
39 }
  
```

Result Details

Result Review

Severity  Enter comment here...

Status

**! Non-terminating loop** ?

The loop is infinite or contains a run-time error.  
loop may fail due to a run-time error (maximum number of iterations: 11)

Result Details

Result Review

Severity  Enter comment here...

Status

**? Illegally dereferenced pointer** ?

Warning: pointer may be outside its bounds  
Dereference of local pointer 'pFlt' (pointer to unsigned int 32, size: 32 bits):  
Pointer is not null.  
Points to 4 bytes at offset multiple of 4 in [0 .. 40] in buffer of 40 bytes, so may be outside bounds.



# Cause of Cruise Control Powering off during fault testing

## Fault Logging

```

18 void fault_log(FAULT_LOG_INFO_T *pFaultInfo)
19 {
20     uint32_t ix;
21
22     uint32_t *pFlt = (uint32_t *)pFaultInfo;
23
24     /* Validate current fault index counter */
25     if (FaultRecIndexCntr >= (MAX_FAULT_LOG_INFO_SIZE - 12u))
26     {
27         FaultRecIndexCntr = 0x0u;
28     }
29
30     /* Store the current fault into circular buffer */
31     for (ix = 0; ix < 12u; ix++)
32     {
33         FaultInfoElement[FaultRecIndexCntr + ix] = *pFlt;
34         *pFlt = 0x0u;
35         pFlt++;
36     }
37     /* Update the circular buffer fault index counter */
38     FaultRecIndexCntr += ix-1;
39 }

```

ix	pFlt	typedef members
0	0x40000000	Expected_Value
1	0x40000004	Received_Value
2	0x40000008	Fault_ID
3	0x4000000c	Fault_Type
4	0x40000010	Time_mSec
5	0x40000014	Time_Sec
6	0x40000018	Time_Min
7	0x4000001c	Time_Hr
8	0x40000020	Additional_Flt_Spec01
9	0x40000024	Additional_Flt_Spec02
10	0x40000028	
11	0x4000002c	CruiseOnOff
		Brake
		CostalSetSw
		AccellResSw
	0x40000030	Speed

# Root cause of Cruise Control Powering off

Fault Logging

```

30 /* Store the current fault into circular buffer */
31 for (ix = 0; ix < 12u; ix++)
32 {
33     FaultInfoElement[FaultRecIndexCntr + ix] = *pFlt;
34     *pFlt = 0x0u;
35     pFlt++;
36 }
41 typedef struct FAULT_LOG_INFO_TAG
42 {
43     uint32_t Expected_Value;
44     uint32_t Received_Value;
45     uint32_t Fault_ID;
46     uint32_t Fault_Type;
47     uint32_t Time_mSec;
48     uint32_t Time_Sec;
49     uint32_t Time_Min;
50     uint32_t Time_Hr;
51     uint32_t Additional_Flt_Spec01;
52     uint32_t Additional_Flt_Spec02;
53     uint32_t Additional_Flt_Spec03;
54     uint32_t Additional_Flt_Spec04;
55 }FAULT_LOG_INFO_T;
56

```

```

30 /* Store the current fault into circular buffer */
31 for (ix = 0; ix < 10u; ix++)
32 {
33     FaultInfoElement[FaultRecIndexCntr + ix] = *pFlt;
34     *pFlt = 0x0u;
35     pFlt++;
36 }
24 /* NEW: Fault log record information */
25 typedef struct FAULT_LOG_INFO_TAG
26 {
27     uint32_t Expected_Value;
28     uint32_t Received_Value;
29     uint32_t Fault_ID;
30     uint32_t Fault_Type;
31     uint32_t Time_mSec;
32     uint32_t Time_Sec;
33     uint32_t Time_Min;
34     uint32_t Time_Hr;
35     uint32_t Additional_Flt_Spec01;
36     uint32_t Additional_Flt_Spec02;
37 }FAULT_LOG_INFO_T;

```

# Fix and verify the hand code is free of Runtime Errors

## Fault Logging

```

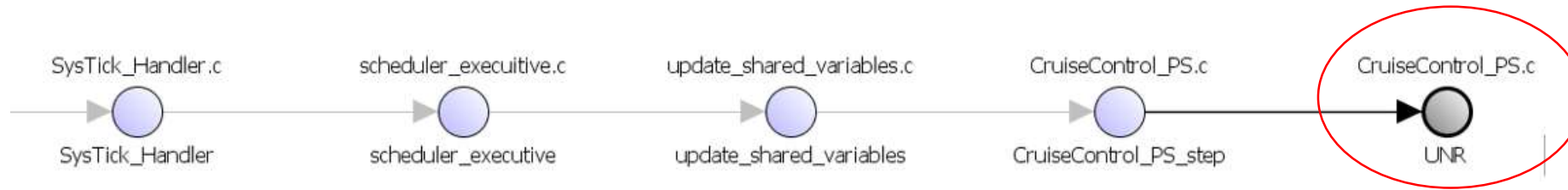
18 void fault_log(FAULT_LOG_INFO_T *pFaultInfo)
19 {
20     uint32_t ix;
21
22     uint32_t *pFlt = (uint32_t *)pFaultInfo;
23
24     /* Validate current fault index counter */
25     if (FaultRecIndexCntr >= (MAX_FAULT_LOG_INFO_SIZE - 12u))
26     {
27         FaultRecIndexCntr = 0x0u;
28     }
29
30     /* Store the current fault into circular buffer */
31     for (ix = 0; ix < 10u; ix++)
32     {
33         FaultInfoElement[FaultRecIndexCntr + ix] = *pFlt;
34         *pFlt = 0x0u;
35         pFlt++;
36     }
37     /* Update the circular buffer fault index counter */
38     FaultRecIndexCntr += ix-1;
39 }

```

ix	pFlt	typedef members
0	0x40000000	Expected_Value
1	0x40000004	Received_Value
2	0x40000008	Fault_ID
3	0x4000000c	Fault_Type
4	0x40000010	Time_mSec
5	0x40000014	Time_Sec
6	0x40000018	Time_Min
7	0x4000001c	Time_Hr
8	0x40000020	Additional_Flt_Spec01
9	0x40000024	Additional_Flt_Spec02
	0x4000002c	CruiseOnOff
		Brake
		CostalSetSw
		AccellResSw
	0x40000030	Speed

# Code Integration Check with Polyspace: Dead Code Found in Generated Code

Target Speed  
Control Module



Vehicle speed signal propagated to  
"CruiseControl\_PS.c" [0 ... 40]

Maximum target speed = 90

```

111 } else if (Speed > maxtspeed) {
112     /* Transition: '<S1>:114' */
113     /* Exit Internal 'ON': '<S1>:54' */
114     DW.is_ON = CruiseContro_IN_NO_ACTIVE_CHILD;
115     DW.is_CRUISE = CruiseControl_PS_IN_STANDBY;
116
117     /* Entry 'STANDBY': '<S1>:52' */
118     engaged = false;

```

Unreachable/Dead code

# Root Cause for Dead Code: Speed Sensor Input Hand Code

Read Inputs

Changing analog-to-digital converter from 14 to 12-bit results in dead code

```

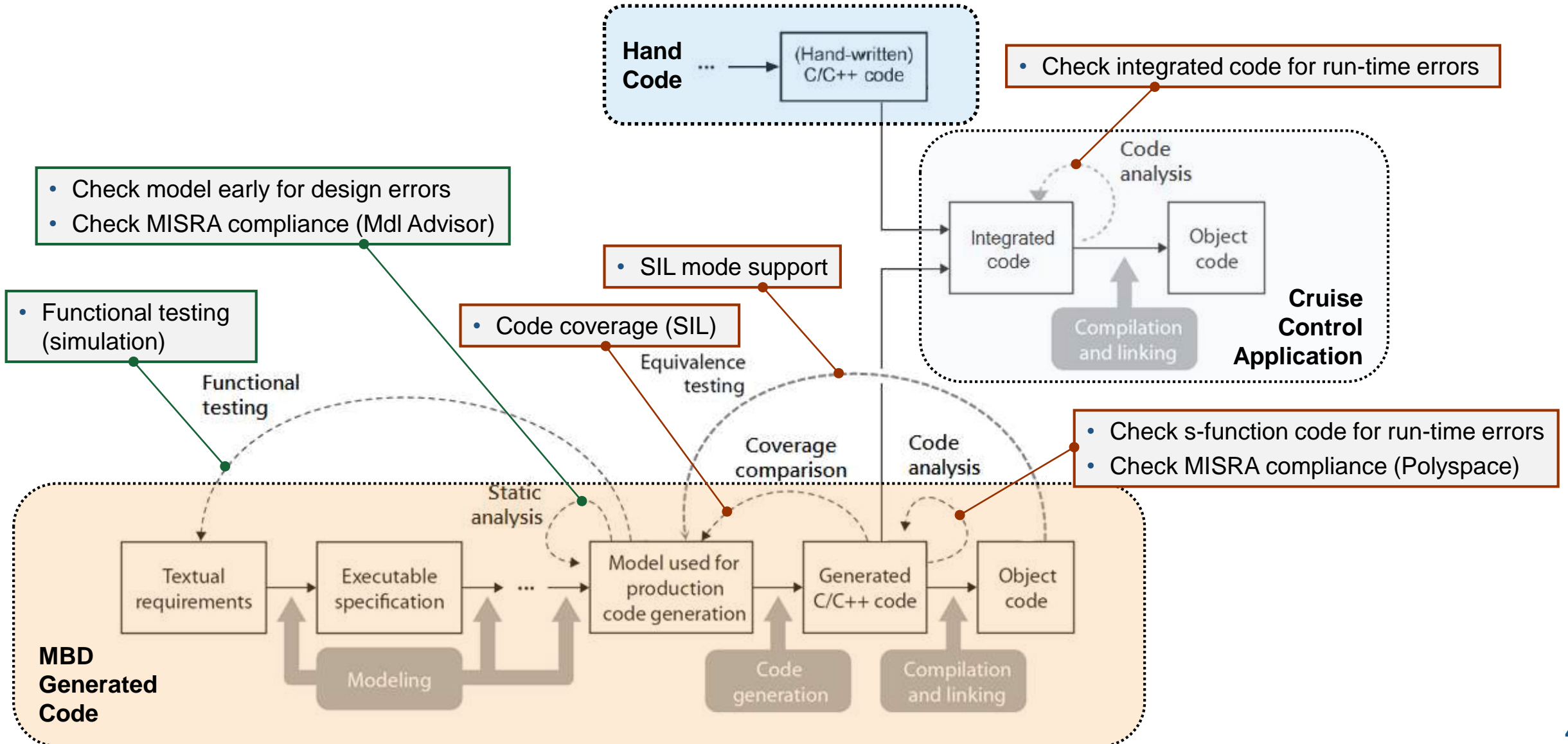
analog.h
19 /* Conversion factors of speed */
20 #define NEW_ECU
21 #ifndef NEW_ECU
22     #define SPEED_MASK 0xFF  /* New ECU */
23 #else
24     #define SPEED_MASK 0x3FFF /* Original design specification */
25 #endif
26
27 /* Scaling for conversion factor for translating sensor input to miles/hr */
28 #define CONV_FACTOR 0.01 /* FAILS */
29
30 #define MAX_AI_RAW_COUNTS_BUFFER_SIZE 100
31
32 AI_Speed.Average = (AI_Speed.Raw_Counts / MAX_AI_RAW_COUNTS_BUFFER_SIZE);
33
34 /* Convert raw counts to speed */
35 AI_Speed.Speed = ((AI_Speed.Average & SPEED_MASK) * CONV_FACTOR);
36
37 /* Updated analog inputs */
38 MDB_Shared_Data.Speed = AI_Speed.Speed;
39 }
    
```

MASK – accounts for scaling down for new ADC from 14-bit to 12-bit

Overlooked changing CONV\_FACTOR for new ADC



# Workflow Summary: Complementary Model & Code Verification



# A Complementary Model and Code Verification Process ...

- ✓ Model and code checks before functional testing to minimize rework
- ✓ Perform functional, dynamic testing with model and code structural analysis with automation, and reuse of test assets
- ✓ Analyze the code to find issues resulting from the integration of
  - hand code
  - s-function code
  - model-based generated code
- ✓ Includes formal methods analysis to go beyond functional testing
- ✓ Enables more, early testing of the model and code
- ✓ Continual increase in design confidence

# Thank You!