Find Defects from Simulink

Introduction

In this tutorial, you analyze the generated code from a Simulink® model using Polyspace® Bug Finder™. To do this analysis, the tutorial follows a common workflow for model-generated code analysis:

1. Create model.
2. Generate code.
3. Select Polyspace configuration options.
4. Run analysis.
5. Review results.

Open Model and Generate Code

1. In MATLAB®, open the Polyspace example model.

```
psdemo_model_link_sl
```

2. Right-click the controller subsystem and select **C/C++ Code** > **Build This Subsystem** to generate code for the controller subsystem.
3. In the Build code for Subsystem: controller window, select **Build.**

Note: The code generation options for this model are already set. For configuring your model, see *Recommended Model Settings for Code Analysis.*

Set Polyspace Options and Run Analysis

1. After the model has finished building, select **Code** > **Polyspace** > **Options.**
2. In the Configuration Parameters window that opens, on the **Polyspace** pane, set the following options.

Option	Value
Product mode	`Bug Finder`
Settings from	`Project configuration and MISRA C 2012 checking for generated code`

These options set the type of Polyspace analysis and configure the analysis to check for bugs and MISRA C® coding rule violations.

3. Apply your changes and close the Configuration Parameters window.

4. Right-click the controller subsystem and select **Polyspace > Verify Code Generated For > Selected Subsystem.**

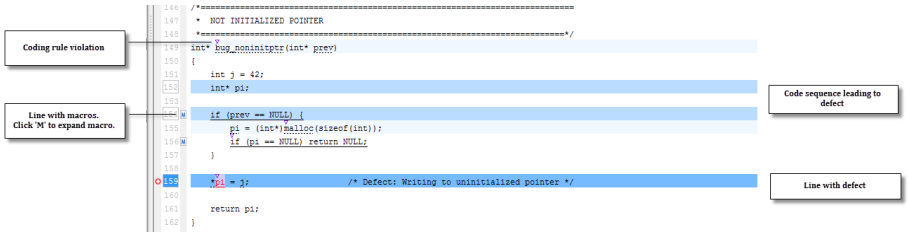
5. You can follow the progress of the analysis in the Command Window.

Review Results

After the analysis has finished, open and review the results in the Polyspace interface.

1. If the results do not open automatically, right-click the controller subsystem and select **Polyspace > Open Results.**

2. Select the **Integer division by zero** result.
   In Bug Finder, you see three windows:

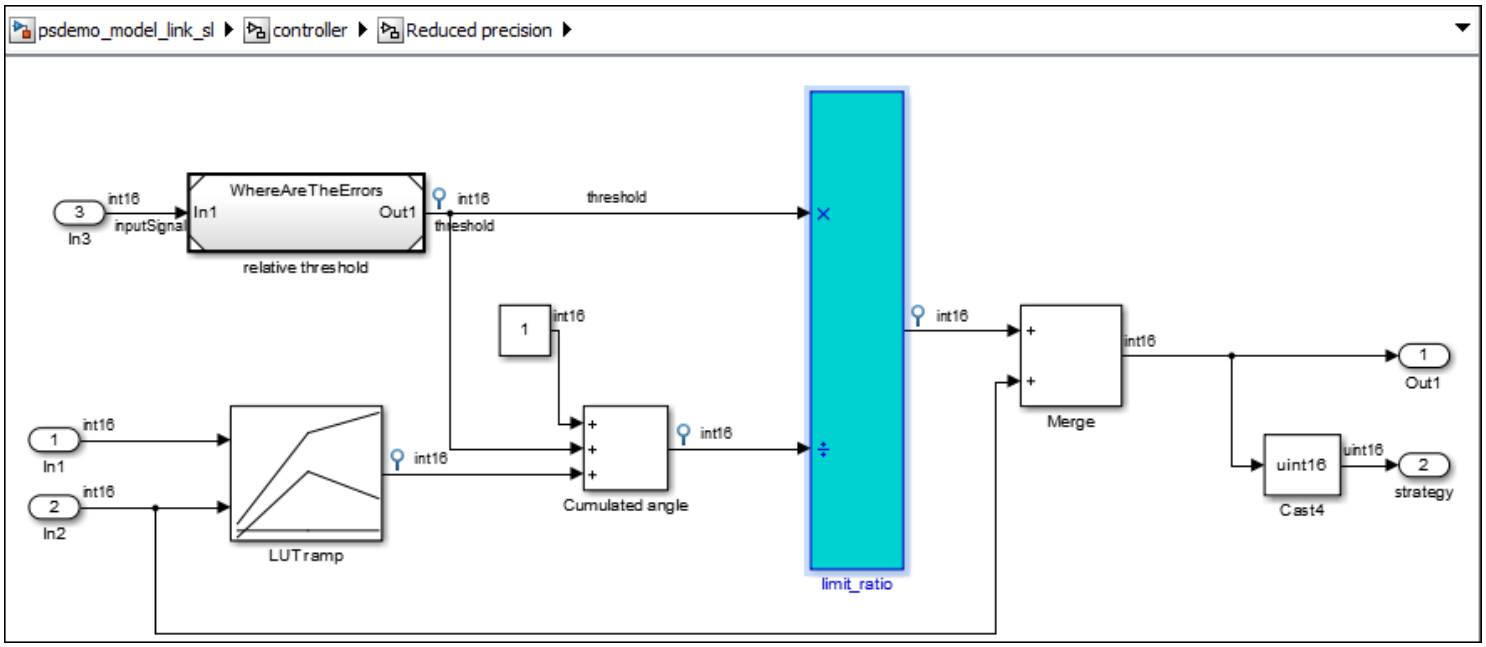
Pane Name	Purpose
Results List	List of results Using the  button, you can view the results ungrouped, by family of defect types, or by file. In each of these views, you can sort or filter the results using the icon in the column headers. Right-click the column header to add or remove columns. You can enter defect-specific comments and justifications by using the **Status** and **Comments** columns.
Result Details	Details about the selected result. The pane includes a description of the result and, if applicable, an event list to help find the root cause of the result. As you review your results, use the **Result Review** section to add justification comments and a status to the result. If you need more help, use the  icon to open documentation about the result.
Source	See the selected defect in the source code. By default, a **Dashboard** tab appears showing results statistics. These statistics can provide a big picture of the defect distribution and top coding rule violations. As you select different checks, the source files open. Results are marked in the source code.  • Red, underlined code with a red empty circle in the margin indicates a defect. • A purple triangle above the code indicates a coding rule violation. • Macro expansions are labeled with a blue M, which toggles between the macro and the executed code.z The line of code with the selected defect is highlighted in dark blue. Related lines of code are highlighted in light blue and a box outlines their line numbers.

3. Look at the **Result Details** pane. The description states that the divisor, `controller _ B.Cumulatedangle`, may be zero.
4. Hover over the red division symbol, **/**, in the **Integer division by zero** line. The tooltip tells you the ranges of the two operands and the result. The range of the divisor (right operand) contains zero. The possibility of this value causes the **Integer division by zero** defect.
5. In the **Result Details** pane, starting at the bottom of the list of events, select the events to see where the value of `controller _ B.Cumulatedangle` is assigned.
   The second event shows that `controller _ B.Cumulatedangle` is assigned the value of tmp. The statements that assign `tmp a non-zero` value are in dead if/else branches.

Fix Model and Rerun Analysis

This section shows you how to fix the **Integer division by zero** defect you examined before. As you learned from reviewing the code, the defect occurs because `controller _ B.Cumulatedangle`, the divisor, can be zero.

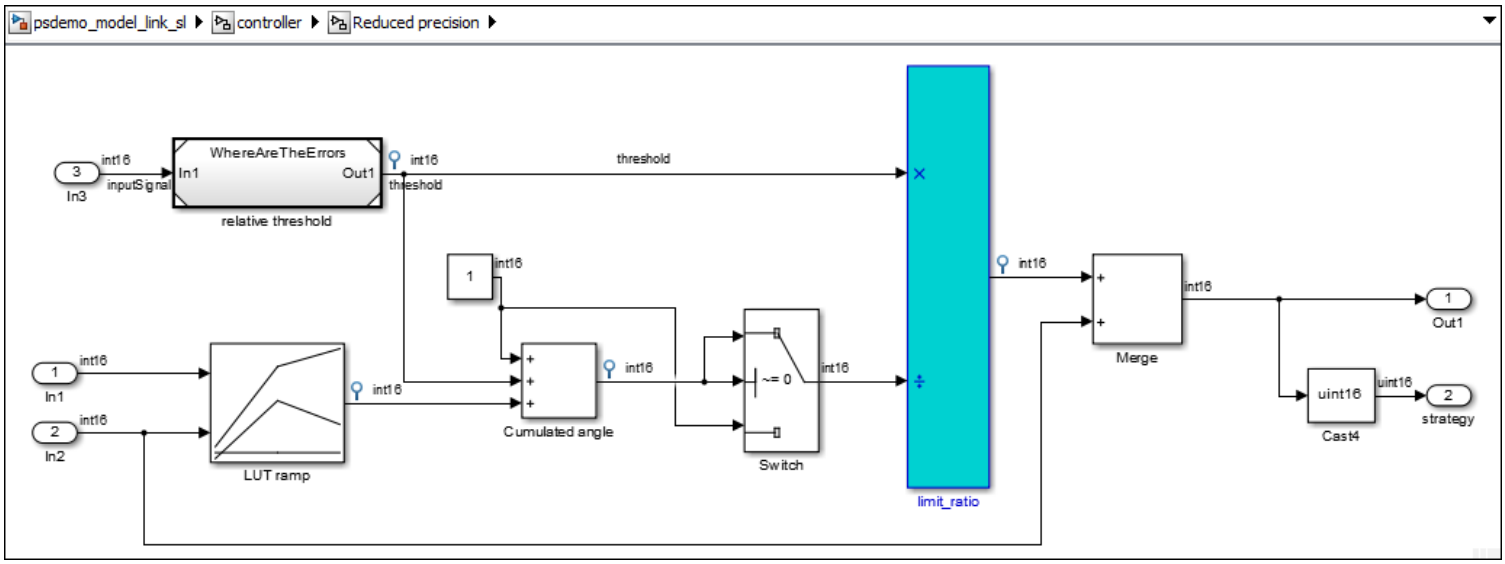
1. To find where this division takes place in the model, in the **Source** pane above the Integer division by zero defect, click `<S4>/limit _ ratio`.
   The `limit _ ratio` block associated with this line of code is highlighted in the Simulink model in blue.



This link between the model and the generated code helps you identify the parts of your model that are causing defect. That way you can fix the defects in the model rather than the code.

2. In between the highlighted `limit _ ratio` block and the Cumulated angle block, add a *Switch* block.
3. Change the criteria property of the Switch block to `u2 ~= 0`.

4. Connect the Switch block to the **`Cumulated angle`** signal and the constant 1 block so that the **`Cumulated angle`** is compared to zero. If the angle is not zero, the cumulated angle is output. If the signal is equal to zero, the 1 is output.
   `The Reduced precision` subsystem should look like the following:



5. Regenerate the code for the **`controller`** subsystem. You might need to save a new version of the model.
6. Rerun the Polyspace analysis for the **`controller`** subsystem.
7. Open the results.
   The new results no longer contain an **Integer division by zero** defect because you fixed the model and by extension the generated code.