MATLAB EXPO

효율적인 모델 기반 설계를 위한 최적화 코드 생성

김학범, MathWorks Korea

MathWorks®

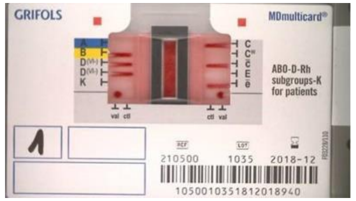
Code Generation Utilized in Various Applications and Industries

The new XC 90 is build on **SPA platform** utilizing Model-Based Design and **AUTOSAR** in Volvo

Alenia Aermacchi develops autopilot software for **DO-178B level A** certification

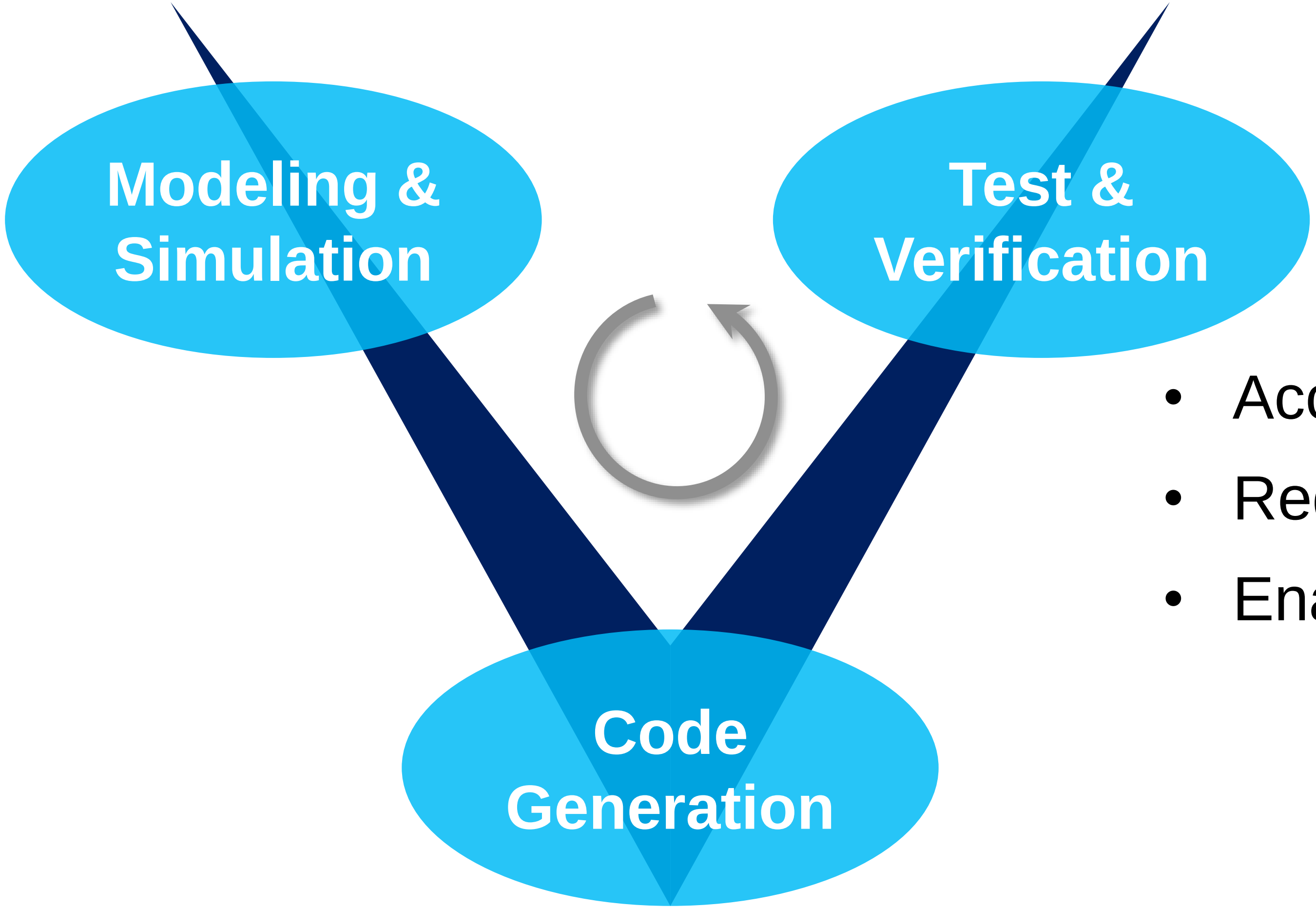
ITK engineering develops **IEC 62304** compliant controller for dental drill motor with MBD

Stem accelerates development of **power electronics control** system with MBD



IDNEO develops embedded **computer vision and machine learning** algorithms for interpreting blood type results

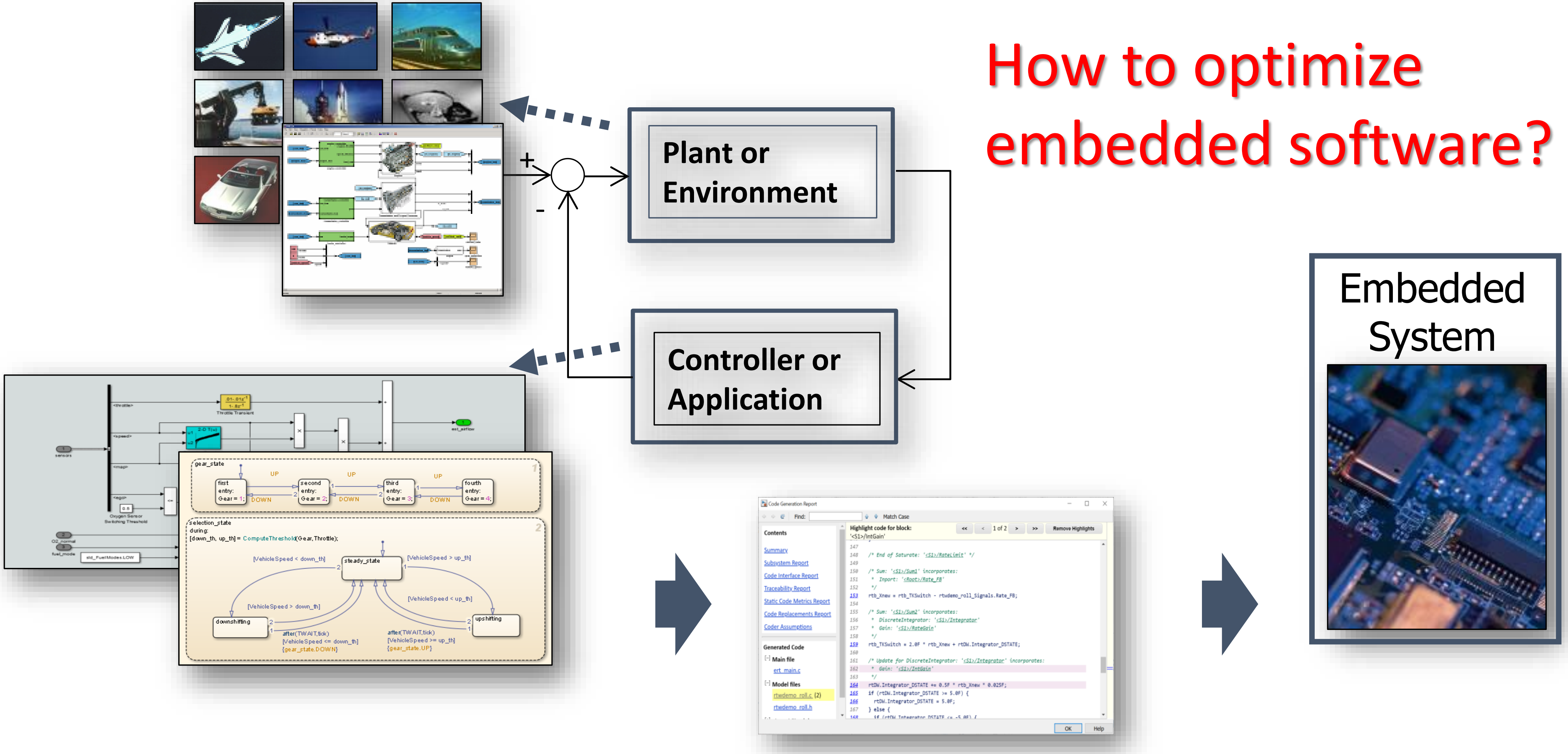
Code Generation Connects Model-Based Design Workflows



- Accelerate development process
- Reduce translation error
- Enable rapid iterative workflows

Design an Embedded Controller

How to optimize embedded software?



```

/* End of Saturate: '<S1>/RateLimit' */

/* Sum: '<S1>/Sum1' incorporates:
 *  Inport: '<Root>/Rate_FB'
 */
rtb_Xnew = rtb_TKSwitch - rtwdemo_roll_Signals.Rate_FB;

/* Sum: '<S1>/Sum2' incorporates:
 *  DiscreteIntegrator: '<S1>/Integrator'
 *  Gain: '<S1>/RateGain'
 */
rtb_TKSwitch = 2.0F * rtb_Xnew + rtDW.Integrator_DSTATE;

/* Update for DiscreteIntegrator: '<S1>/Integrator' incorporates:
 *  Gain: '<S1>/IntGain'
 */
rtDW.Integrator_DSTATE += 0.5F * rtb_Xnew * 0.025F;
if (rtDW.Integrator_DSTATE >= 5.0F) {
  rtDW.Integrator_DSTATE = 5.0F;
} else {
  if (rtDW.Integrator_DSTATE <= -5.0F) {
```

Approach to Code Efficiency with Model-Based Design

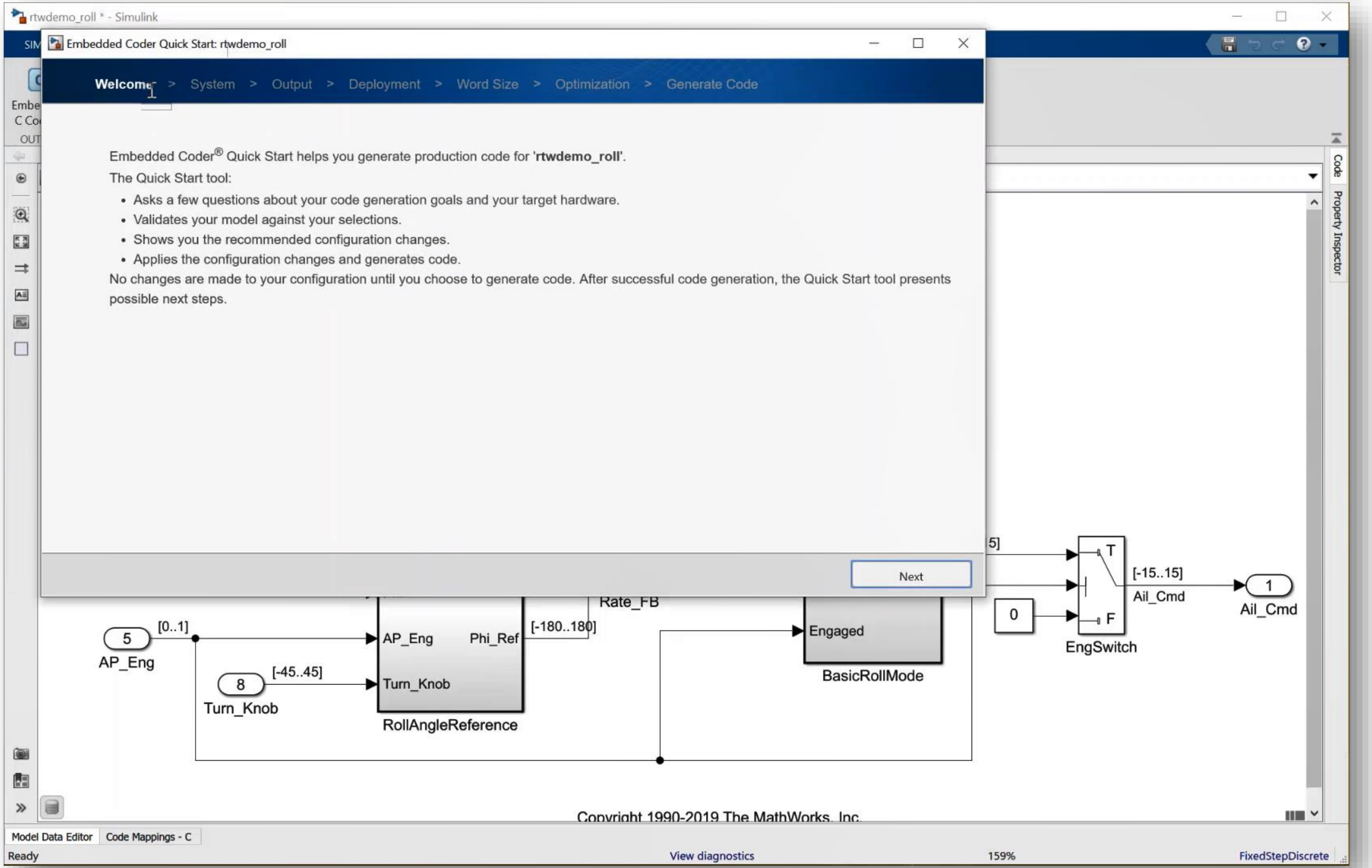
Model Analysis

- Model level & Algorithm level analysis
- Application-aware optimizations (modeling pattern)

Code Generation

- Implementation level analysis
- Target-aware optimizations (resources)

Demo: Embedded Coder Quick Start



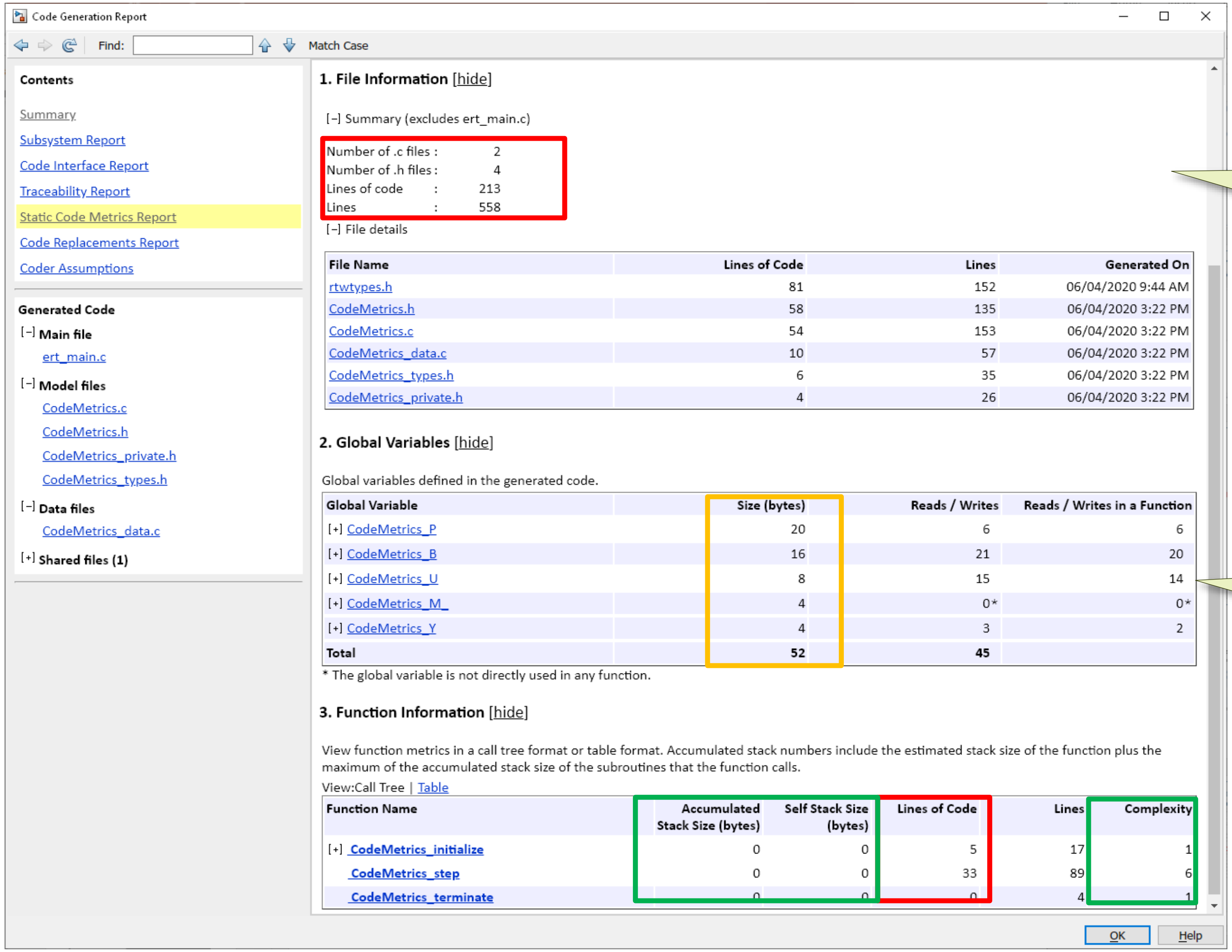
Embedded Coder® Quick Start helps you generate production code for '**rtwdemo_roll**'.

The Quick Start tool:

- Asks a few questions about your code generation goals and your target hardware.
- Validates your model against your selections.
- Shows you the recommended configuration changes.
- Applies the configuration changes and generates code.

No changes are made to your configuration until you choose to generate code. After successful code generation, the Quick Start tool presents possible next steps.

Static Code Metrics Report



Contents

- Summary
- Subsystem Report
- Code Interface Report
- Traceability Report
- Static Code Metrics Report
- Code Replacements Report
- Coder Assumptions

Generated Code

[-] **Main file**

- ert_main.c

[-] **Model files**

- CodeMetrics.c
- CodeMetrics.h
- CodeMetrics_private.h
- CodeMetrics_types.h

[-] **Data files**

- CodeMetrics_data.c

[+] **Shared files (1)**

1. File Information [hide]

[-] Summary (excludes ert_main.c)

Number of .c files	:	2
Number of .h files	:	4
Lines of code	:	213
Lines	:	558

[-] File details

File Name	Lines of Code	Lines	Generated On
rtwtypes.h	81	152	06/04/2020 9:44 AM
CodeMetrics.h	58	135	06/04/2020 3:22 PM
CodeMetrics.c	54	153	06/04/2020 3:22 PM
CodeMetrics_data.c	10	57	06/04/2020 3:22 PM
CodeMetrics_types.h	6	35	06/04/2020 3:22 PM
CodeMetrics_private.h	4	26	06/04/2020 3:22 PM

2. Global Variables [hide]

Global variables defined in the generated code.

Global Variable	Size (bytes)	Reads / Writes	Reads / Writes in a Function
[+] CodeMetrics_P	20	6	6
[+] CodeMetrics_B	16	21	20
[+] CodeMetrics_U	8	15	14
[+] CodeMetrics_M_	4	0*	0*
[+] CodeMetrics_Y	4	3	2
Total	**52**	**45**	

* The global variable is not directly used in any function.

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

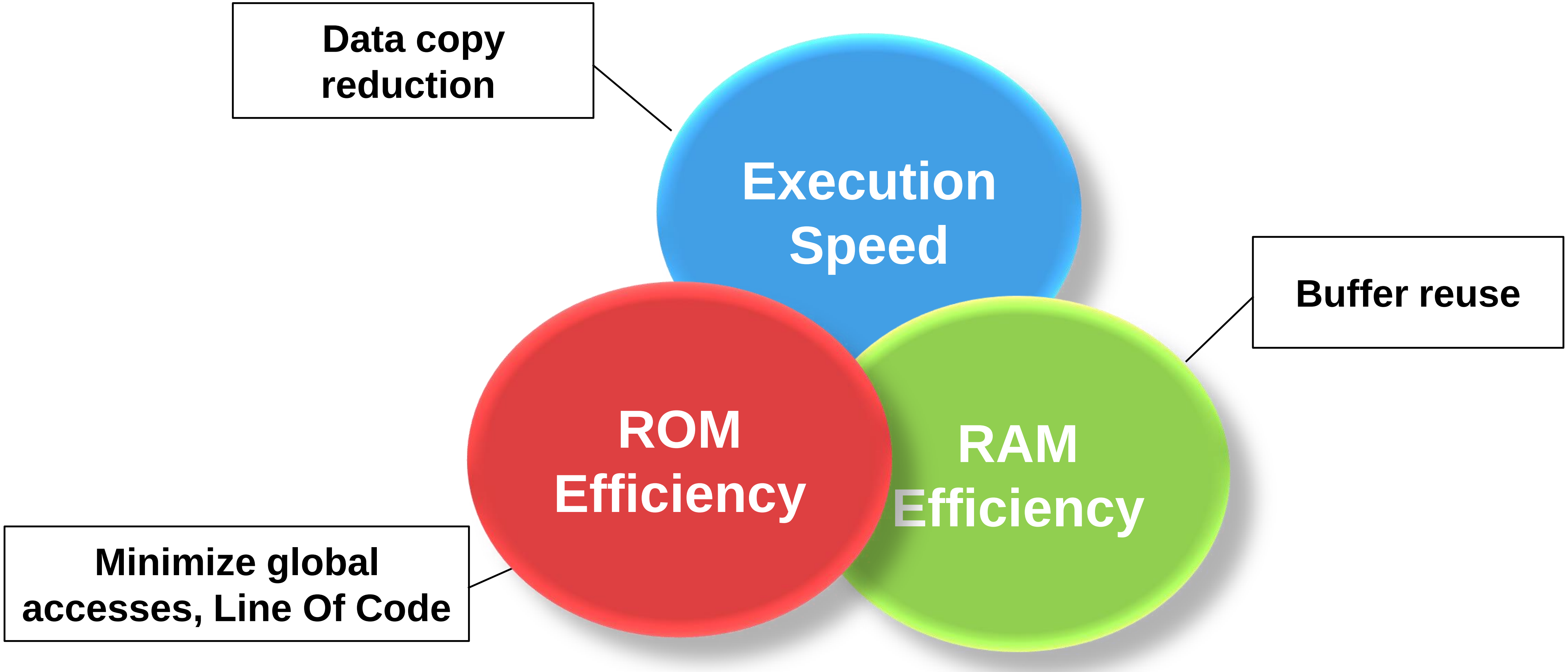
Function Name	Accumulated Stack Size (bytes)	Self Stack Size (bytes)	Lines of Code	Lines	Complexity
[+] CodeMetrics_initialize	0	0	5	17	1
CodeMetrics_step	0	0	33	89	6
CodeMetrics_terminate	0	0	0	4	1

Lines of code

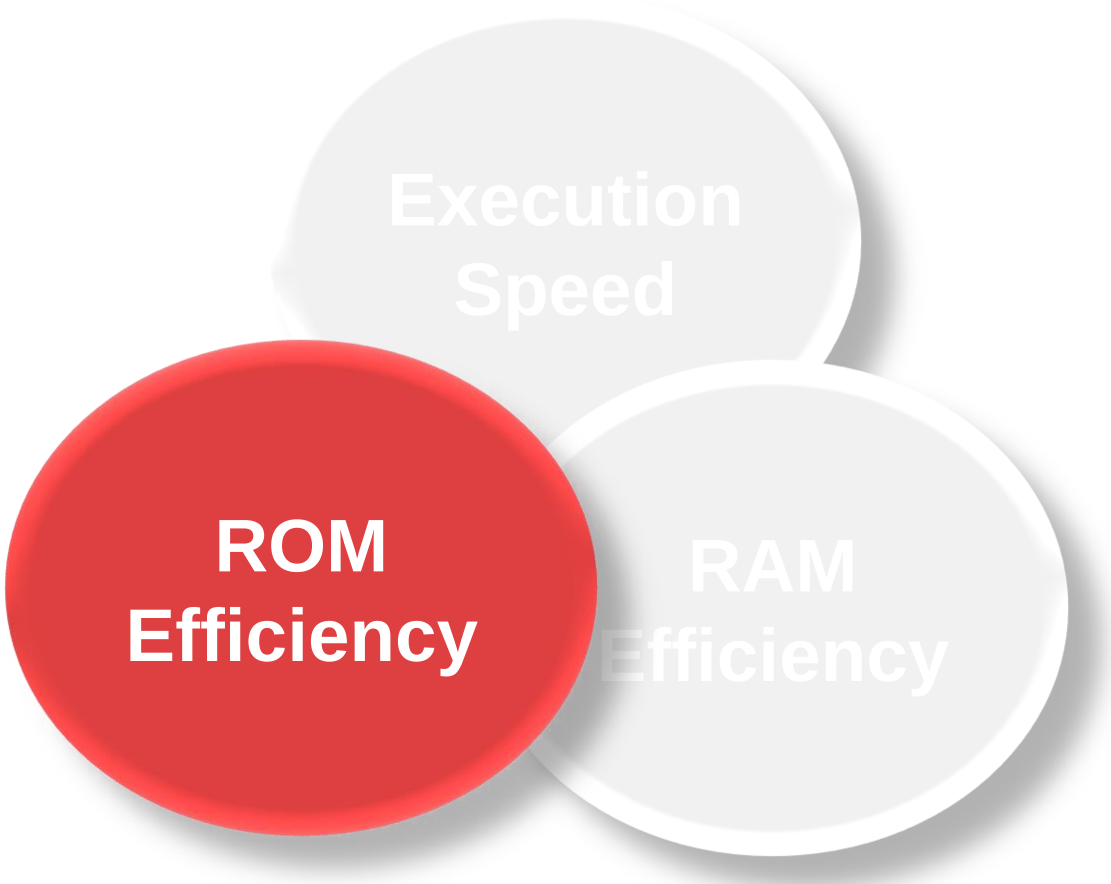
Usage of global variables

Estimated stack size /Cyclomatic complexity

RAM, ROM and Execution Performance



RAM, ROM and Execution Performance

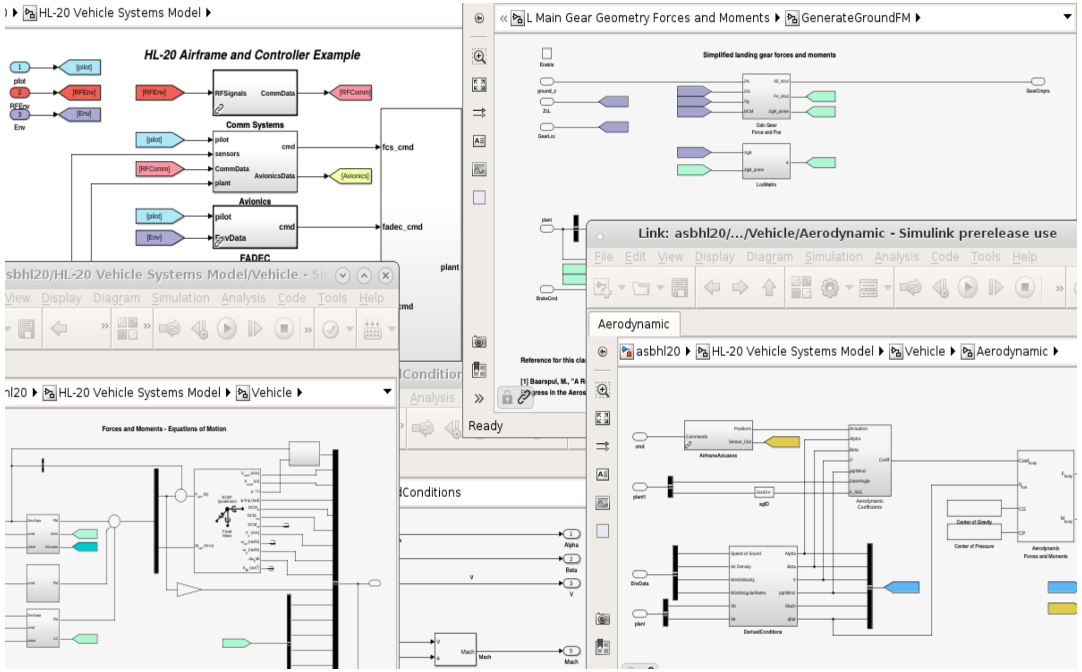


Challenge: Maintaining Large and Complex Systems

ROM Efficiency

- Size and complexity of systems are increasing
  - *“Typical ECU contains 2000 function components that are each developed by a different person”* Automotive customer

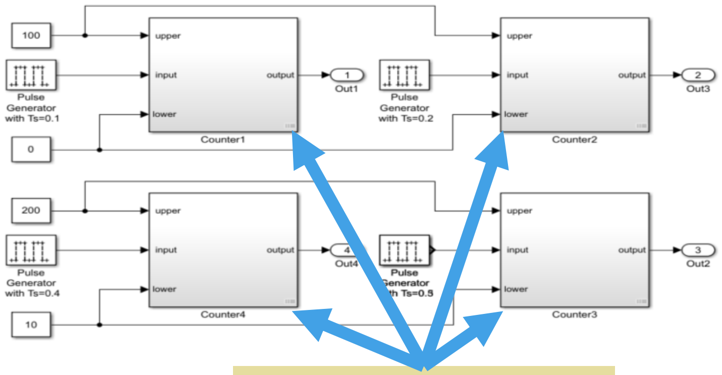
- ”Enforcement of low complexity” required for model standards
  - ISO 26262-6 “Product Development at the Software Level”, Table 1


HL-20 Vehicle Systems Model
HL-20 Airframe and Controller Example
Comm Systems
Avionics
FADEC
fcs_cmd
fadec_cmd
L Main Gear Geometry Forces and Moments ▸ GenerateGroundFM
Simplified landing gear forces and moments
Link: asbhl20/.../Vehicle/Aerodynamic - Simulink prerelease use
File Edit View Display Diagram Simulation Analysis Code Tools Help
Aerodynamic
asbhl20 ▸ HL-20 Vehicle Systems Model ▸ Vehicle ▸ Aerodynamic
sbhl20/HL-20 Vehicle Systems Model/Vehicle
View Display Diagram Simulation Analysis Code Tools Help
hl20 ▸ HL-20 Vehicle Systems Model ▸ Vehicle
Forces and Moments - Equations of Motion
Ready
Conditions

Challenge: Maintaining Large and Complex Systems

ROM Efficiency

- Studies estimate 13-20% of code in large systems are cloned *



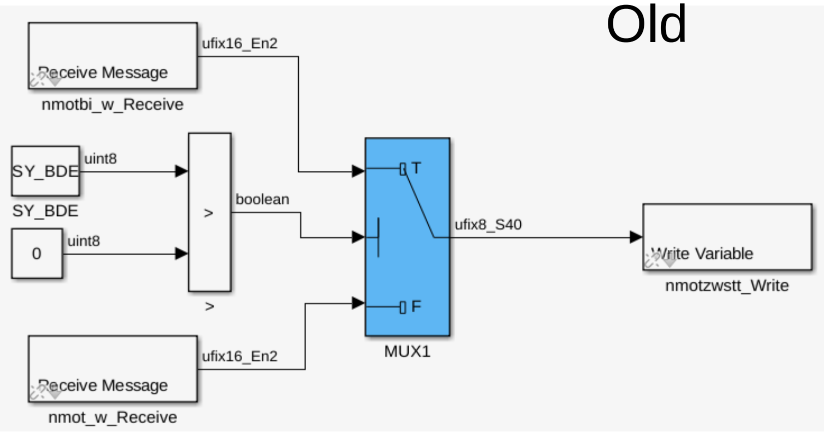
100
Pulse Generator with Ts=0.1
0
upper
input
lower
output
Counter1
Out1
Pulse Generator with Ts=0.2
Counter2
Out3
200
Pulse Generator with Ts=0.4
10
Counter4
Out4
Pulse Generator with Ts=0.5
Counter3
Out2

Copied Subsystems

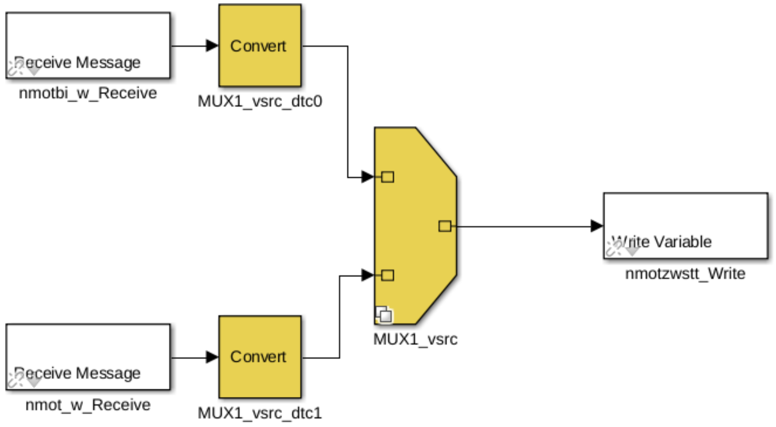
- Old fashion modeling patterns appear:

System constant

Old



New



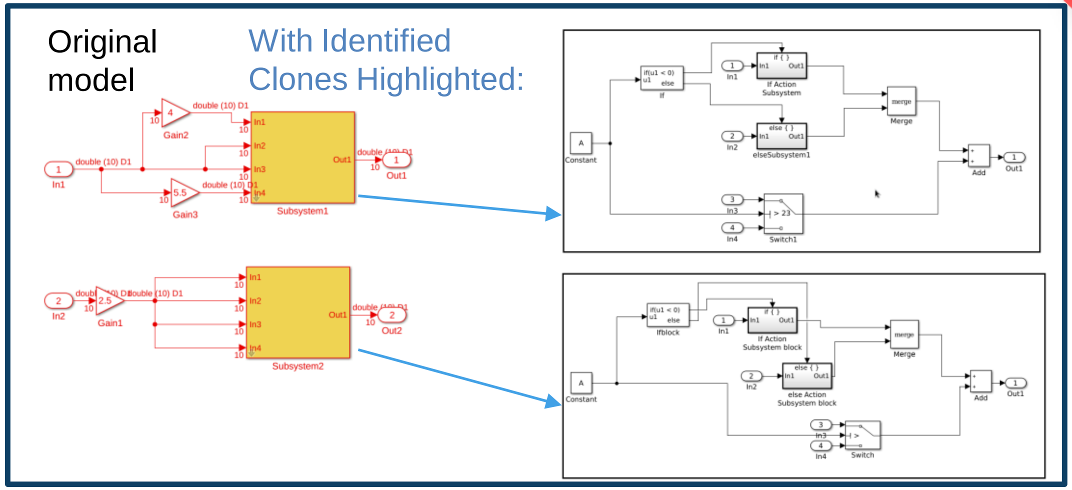
* Source: Roy and Cordy A Survey on Software Clone Detection Research, Sept 2007
Baker. On Finding Duplication and Near-Duplication in Large Software Systems. In Proceedings of the Second Working Conference on Reverse Engineering (WCRE'95), July 1995.

Clone Detection & Refactoring

ROM Efficiency

Clone Detection

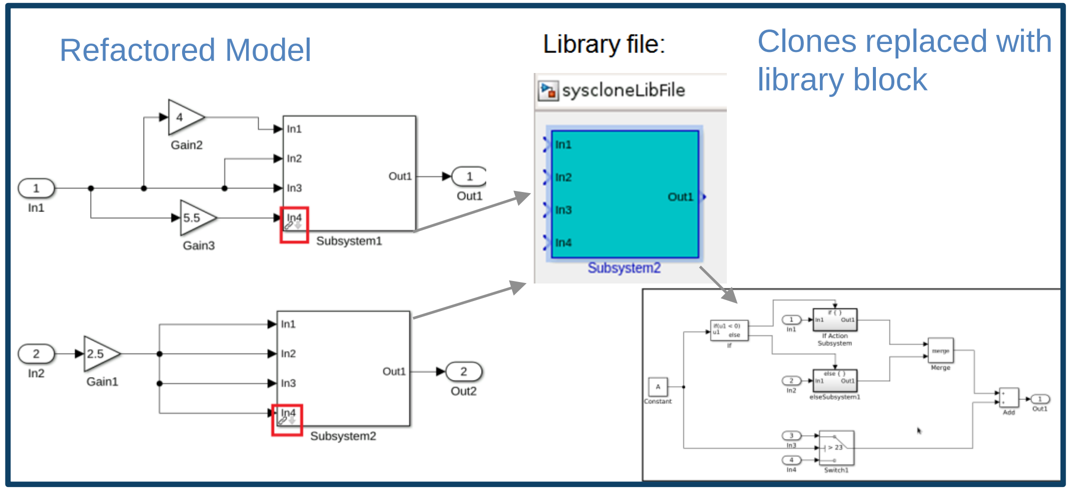
- Find duplicate model content in your design
- Locate opportunities to optimize with a library


Original model

With Identified Clones Highlighted:

Refactoring

- Replace exact clones with library blocks
- Improve reuse and maintainability

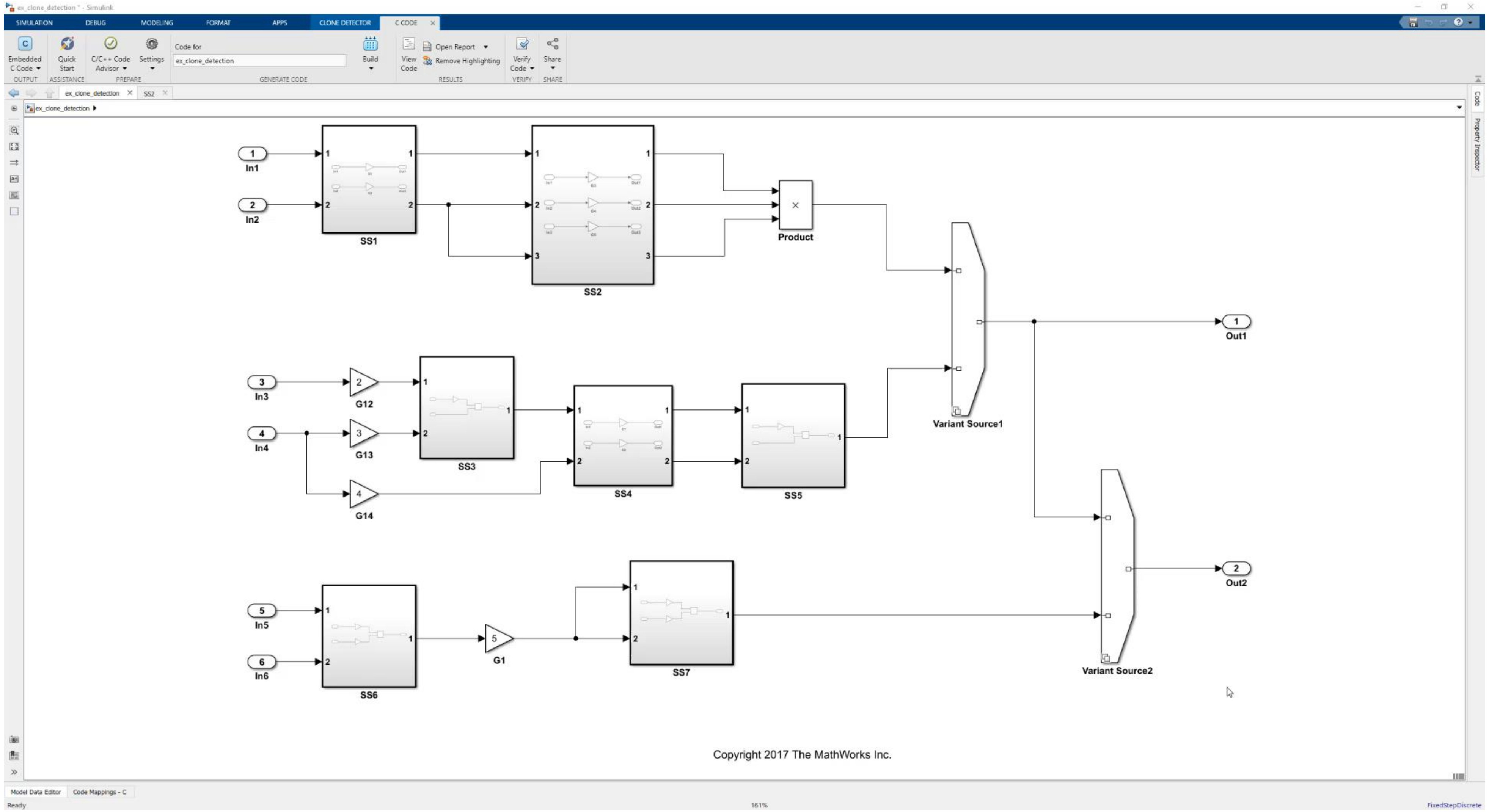

Refactored Model

Library file:

Clones replaced with library block

DEMO: Detect Clone in Model

ROM Efficiency



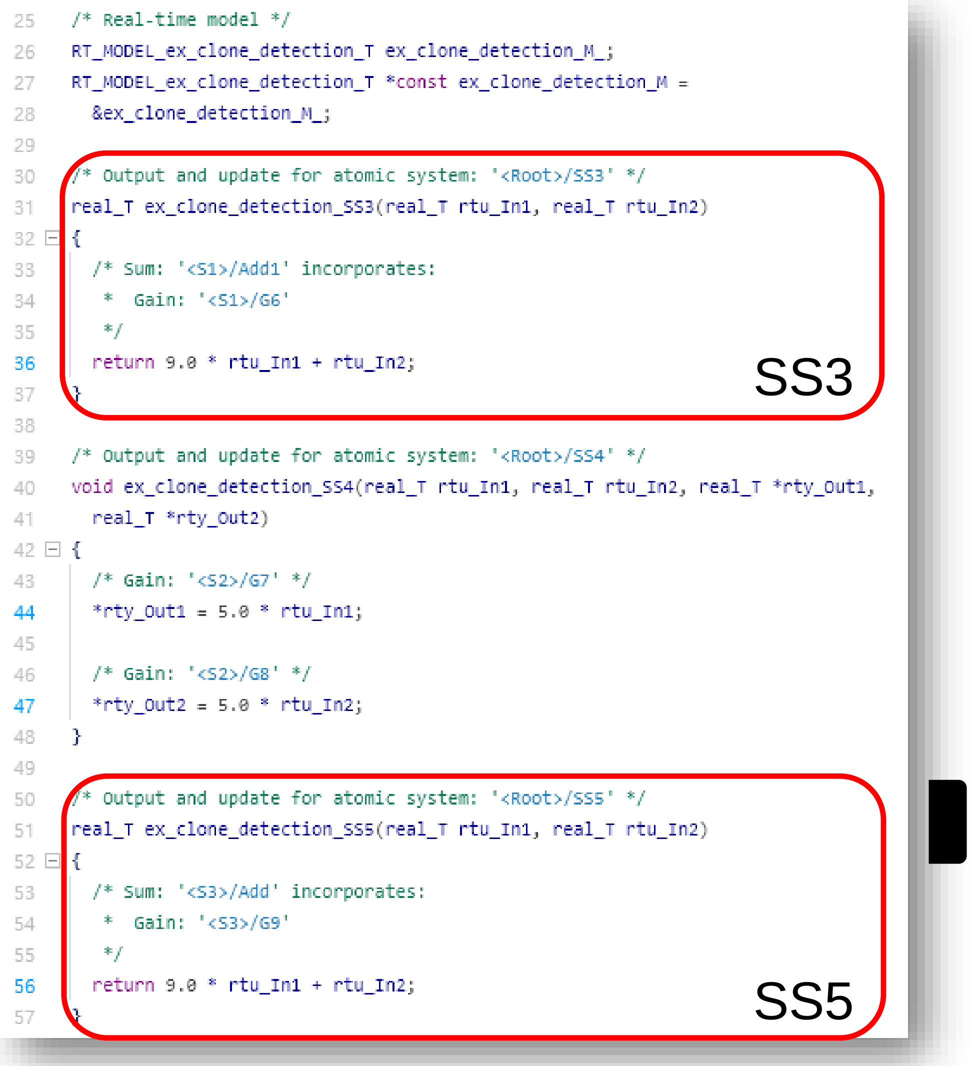
Code for: ex_clone_detection

Copyright 2017 The MathWorks Inc.

Ready 161% FixedStepDiscrete

Review Generated Code

ROM Efficiency



```c
/* Real-time model */
RT_MODEL_ex_clone_detection_T ex_clone_detection_M_;
RT_MODEL_ex_clone_detection_T *const ex_clone_detection_M =
  &ex_clone_detection_M_;

/* Output and update for atomic system: '<Root>/SS3' */
real_T ex_clone_detection_SS3(real_T rtu_In1, real_T rtu_In2)
{
  /* Sum: '<S1>/Add1' incorporates:
   *  Gain: '<S1>/G6'
   */
  return 9.0 * rtu_In1 + rtu_In2;
}

/* Output and update for atomic system: '<Root>/SS4' */
void ex_clone_detection_SS4(real_T rtu_In1, real_T rtu_In2, real_T *rty_Out1,
  real_T *rty_Out2)
{
  /* Gain: '<S2>/G7' */
  *rty_Out1 = 5.0 * rtu_In1;

  /* Gain: '<S2>/G8' */
  *rty_Out2 = 5.0 * rtu_In2;
}

/* Output and update for atomic system: '<Root>/SS5' */
real_T ex_clone_detection_SS5(real_T rtu_In1, real_T rtu_In2)
{
  /* Sum: '<S3>/Add' incorporates:
   *  Gain: '<S3>/G9'
   */
  return 9.0 * rtu_In1 + rtu_In2;
}
```

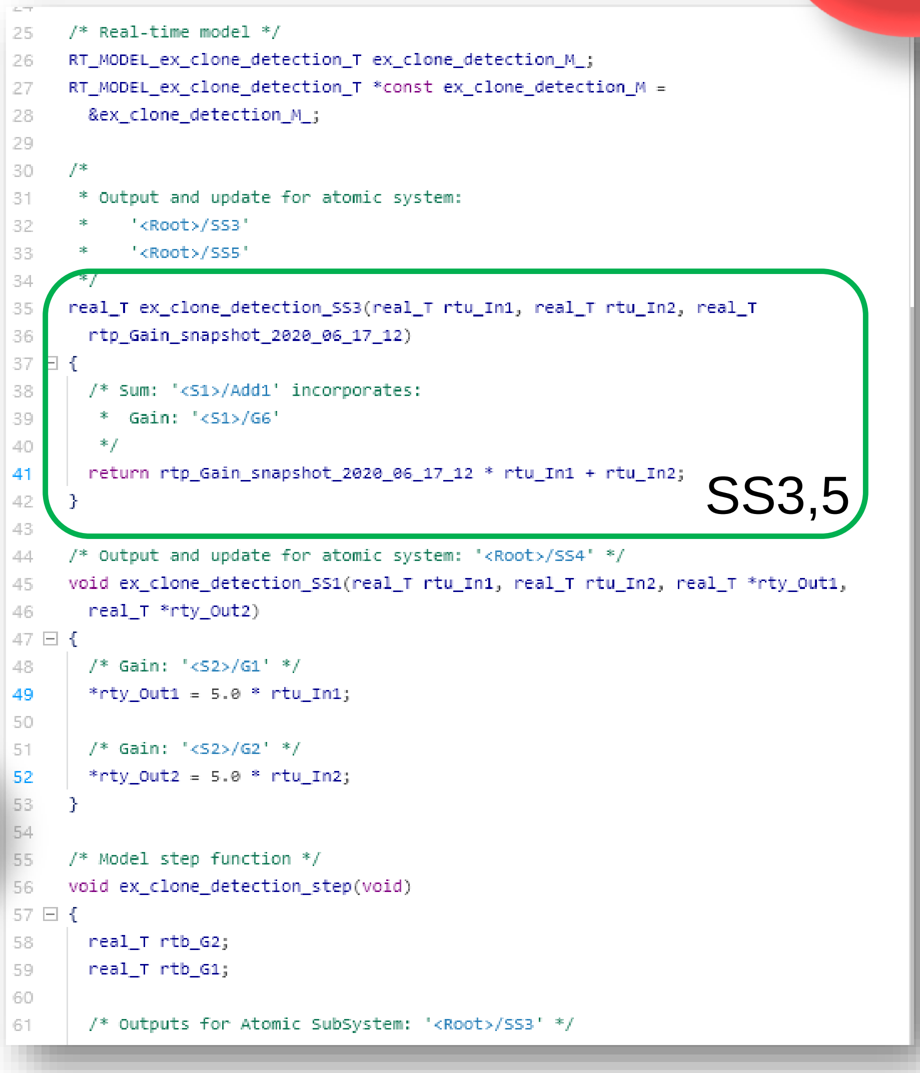
SS3

SS5

Before

Refactoring

ROM Efficiency



```c
/* Real-time model */
RT_MODEL_ex_clone_detection_T ex_clone_detection_M_;
RT_MODEL_ex_clone_detection_T *const ex_clone_detection_M =
  &ex_clone_detection_M_;

/*
 * Output and update for atomic system:
 *    '<Root>/SS3'
 *    '<Root>/SS5'
 */
real_T ex_clone_detection_SS3(real_T rtu_In1, real_T rtu_In2, real_T
  rtp_Gain_snapshot_2020_06_17_12)
{
  /* Sum: '<S1>/Add1' incorporates:
   *  Gain: '<S1>/G6'
   */
  return rtp_Gain_snapshot_2020_06_17_12 * rtu_In1 + rtu_In2;
}

/* Output and update for atomic system: '<Root>/SS4' */
void ex_clone_detection_SS1(real_T rtu_In1, real_T rtu_In2, real_T *rty_Out1,
  real_T *rty_Out2)
{
  /* Gain: '<S2>/G1' */
  *rty_Out1 = 5.0 * rtu_In1;

  /* Gain: '<S2>/G2' */
  *rty_Out2 = 5.0 * rtu_In2;
}

/* Model step function */
void ex_clone_detection_step(void)
{
  real_T rtb_G2;
  real_T rtb_G1;

  /* Outputs for Atomic SubSystem: '<Root>/SS3' */
```

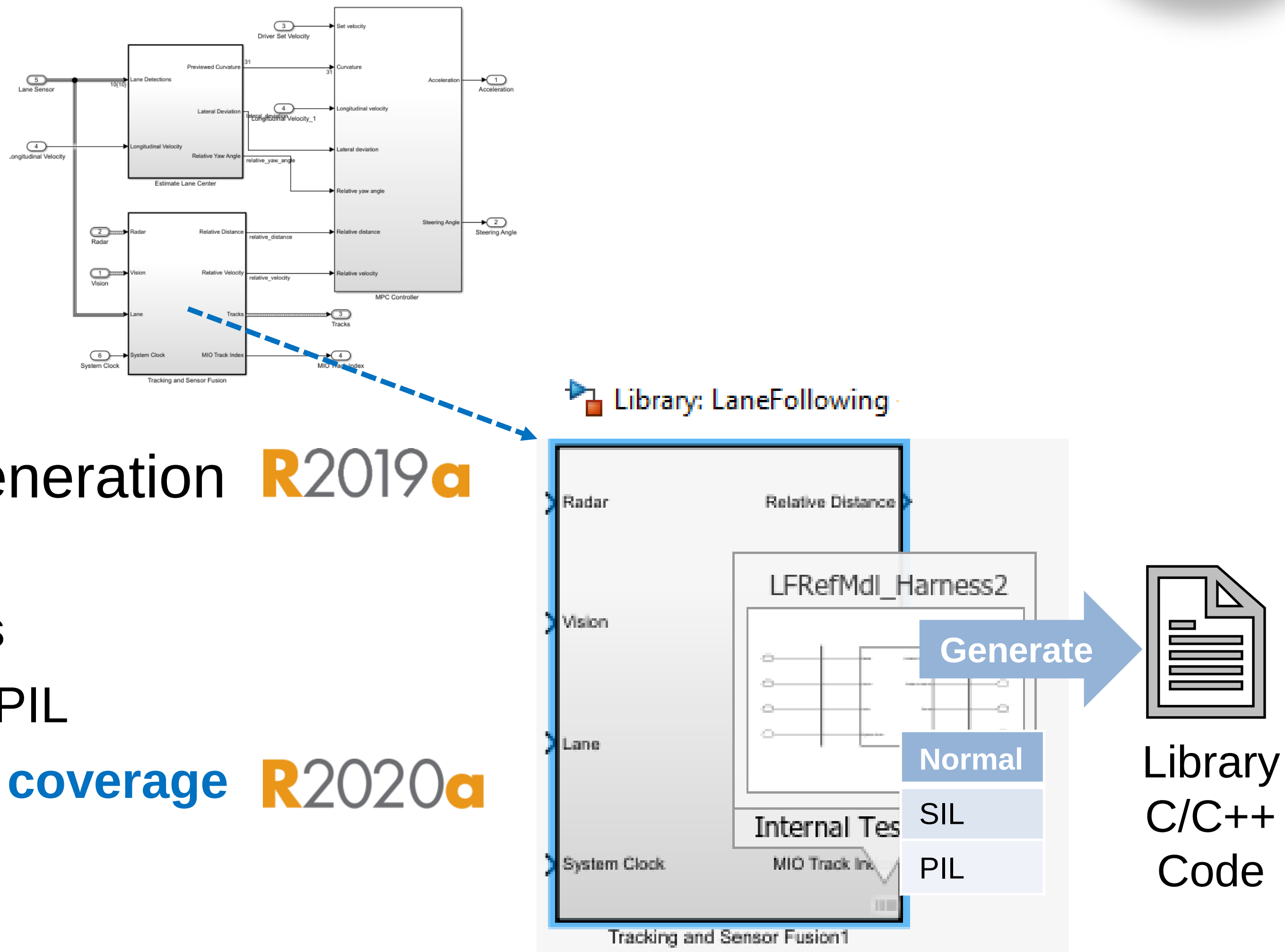
SS3,5

After

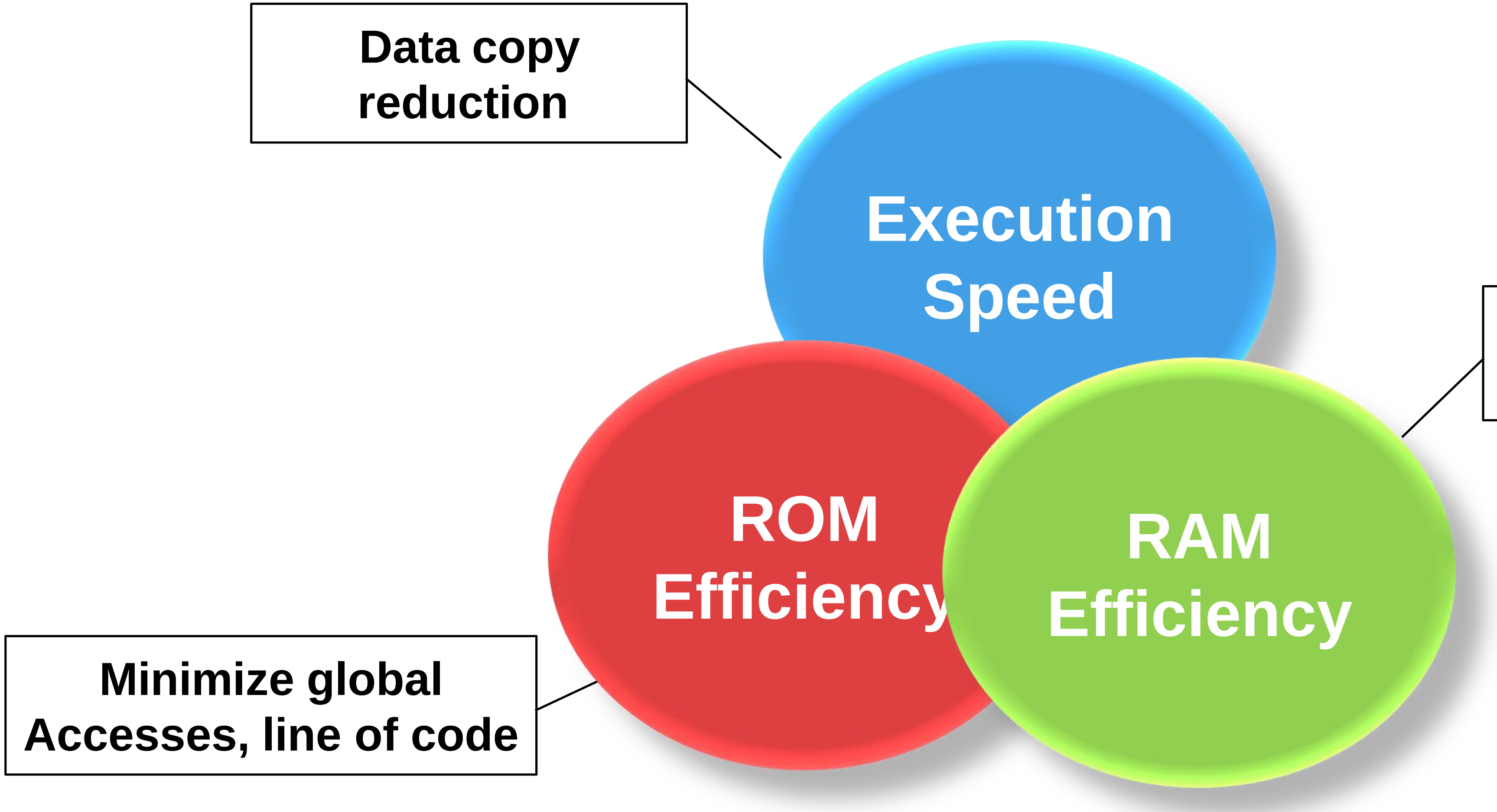
Library-Based Subsystem Code Generation

ROM Efficiency

- System complexity → Unit testing
  - Models ✓
  - Subsystems ?
- Library-based Subsystem Code Generation **R2019a**
  - Lock down function interfaces
  - Generate small reusable sub-functions
  - Verify usage within a model using SIL/PIL
  - **SIL/PIL unit test in library with code coverage** **R2020a**



RAM, ROM and Execution Performance



Data copy reduction

Execution Speed

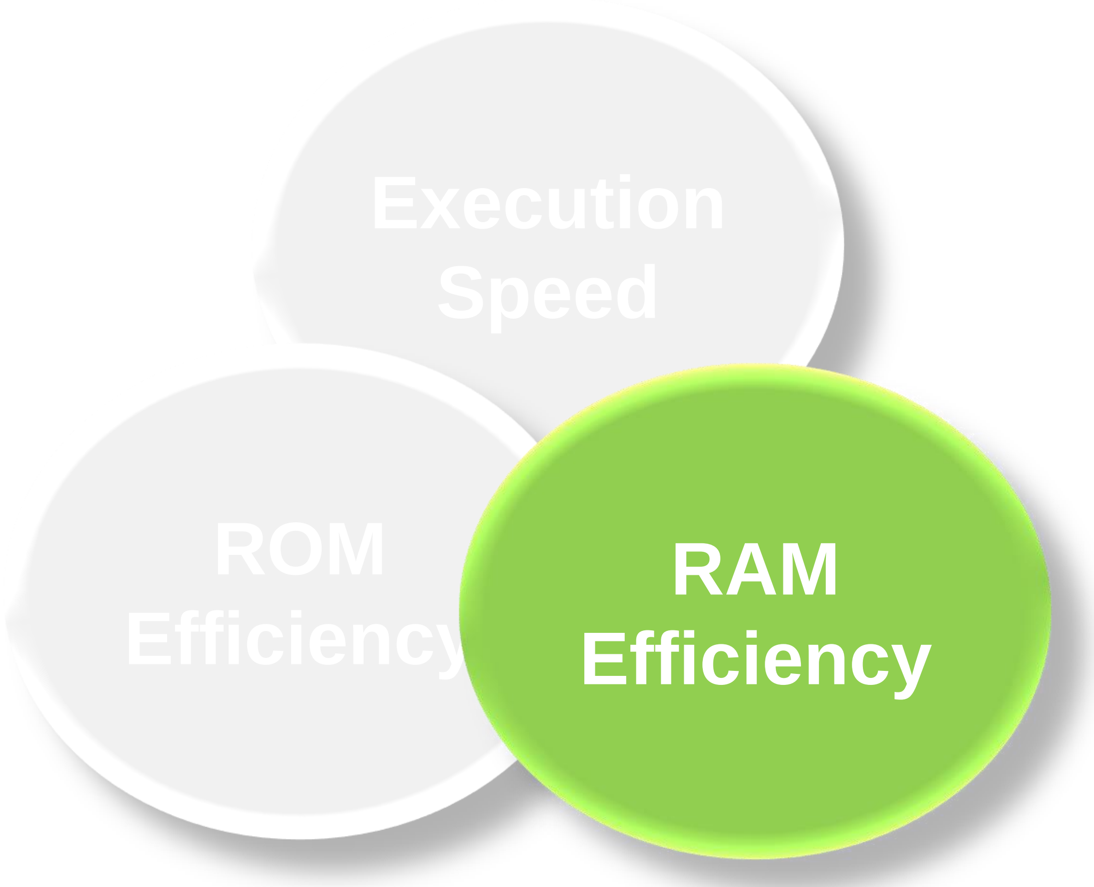
Buffer reuse

ROM Efficiency

RAM Efficiency

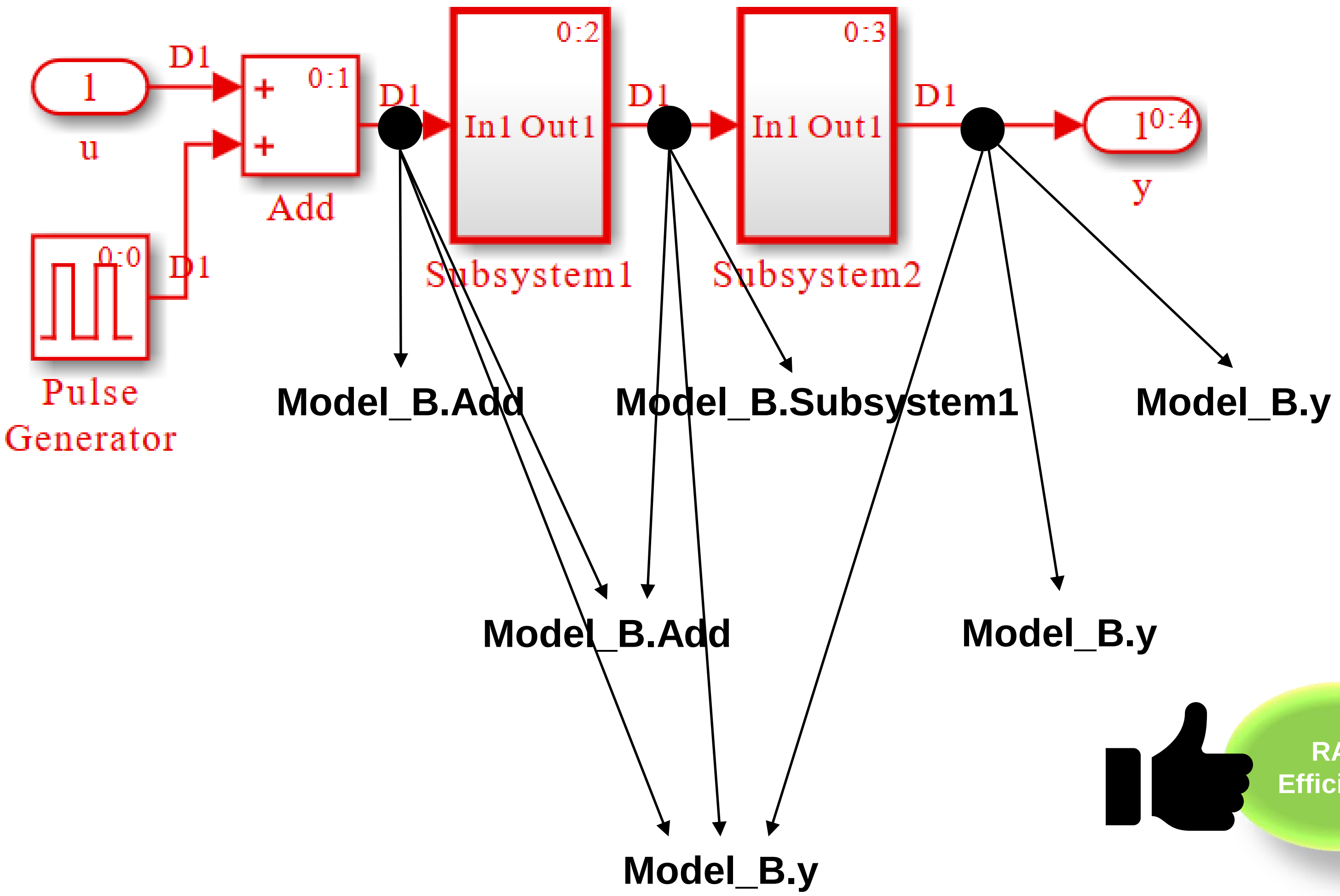
Minimize global Accesses, line of code

RAM, ROM and Execution Performance



Reuse Local Block Output

RAM Efficiency



3 buffers

2 buffers

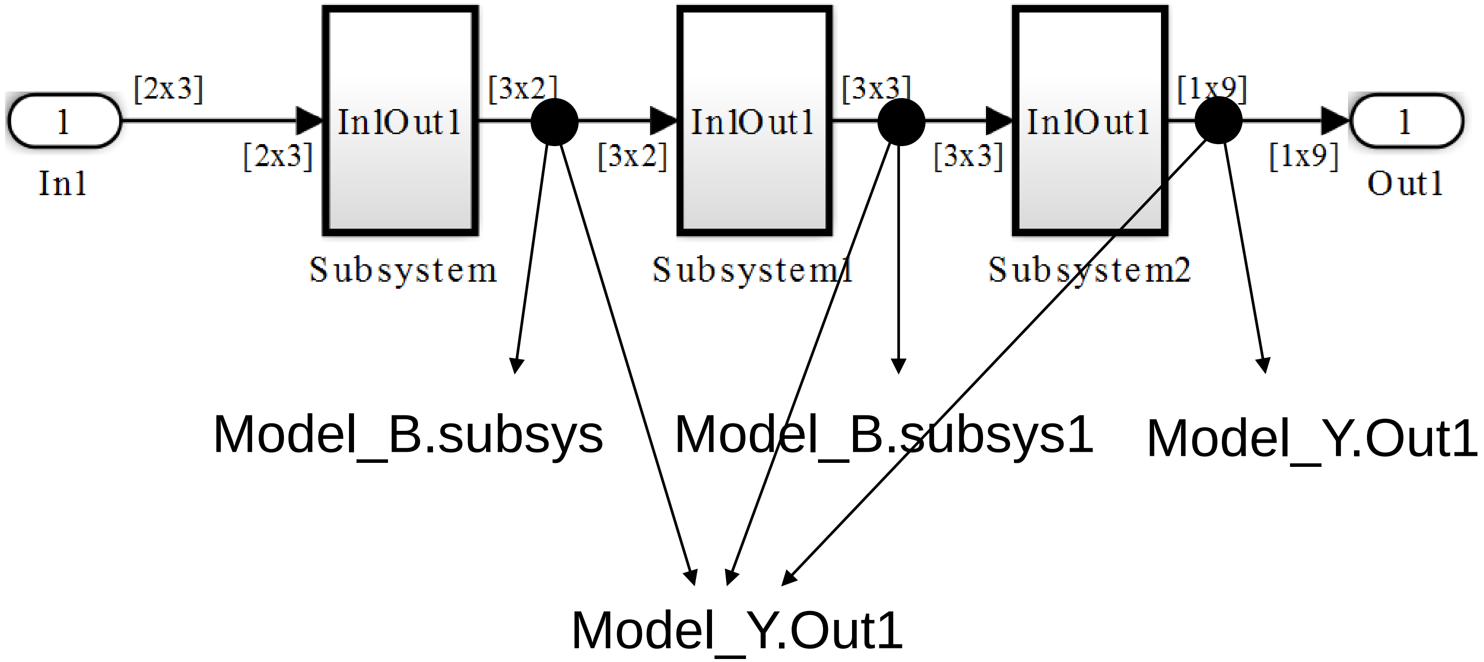
1 buffer

RAM Efficiency

Reuse Local Block Output

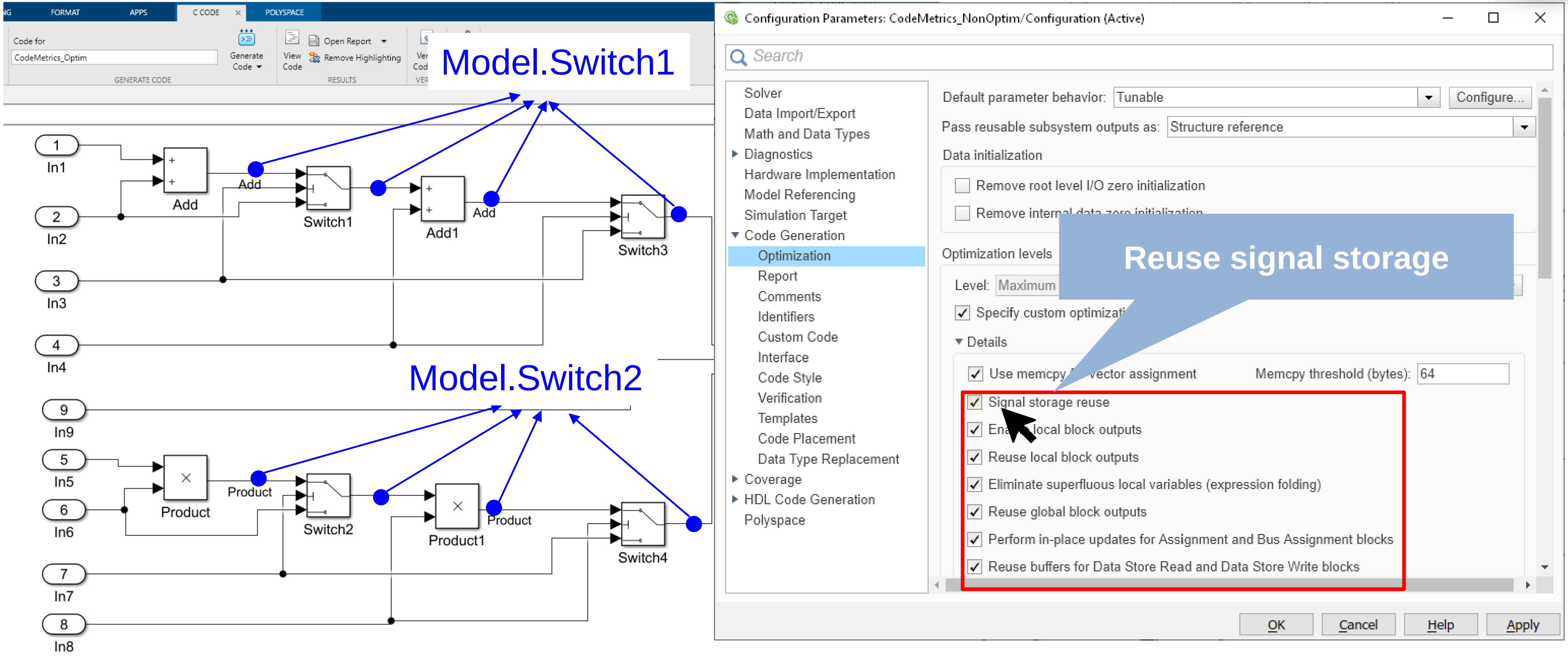
RAM Efficiency

- Reuse buffers with different sizes and/or shapes (dimensions)
  - Different buffers collapse to one, the biggest size is kept



Reuse Local Block Output

RAM Efficiency



Model.Switch1

Model.Switch2

Reuse signal storage

Default parameter behavior: Tunable

Pass reusable subsystem outputs as: Structure reference

Data initialization

☐ Remove root level I/O zero initialization

☐ Remove internal data zero initialization

Optimization levels

Level: Maximum

☑ Specify custom optimizati

Details

☑ Use memcpy for vector assignment

Memcpy threshold (bytes): 64

☑ Signal storage reuse

☑ Ena local block outputs

☑ Reuse local block outputs

☑ Eliminate superfluous local variables (expression folding)

☑ Reuse global block outputs

☑ Perform in-place updates for Assignment and Bus Assignment blocks

☑ Reuse buffers for Data Store Read and Data Store Write blocks

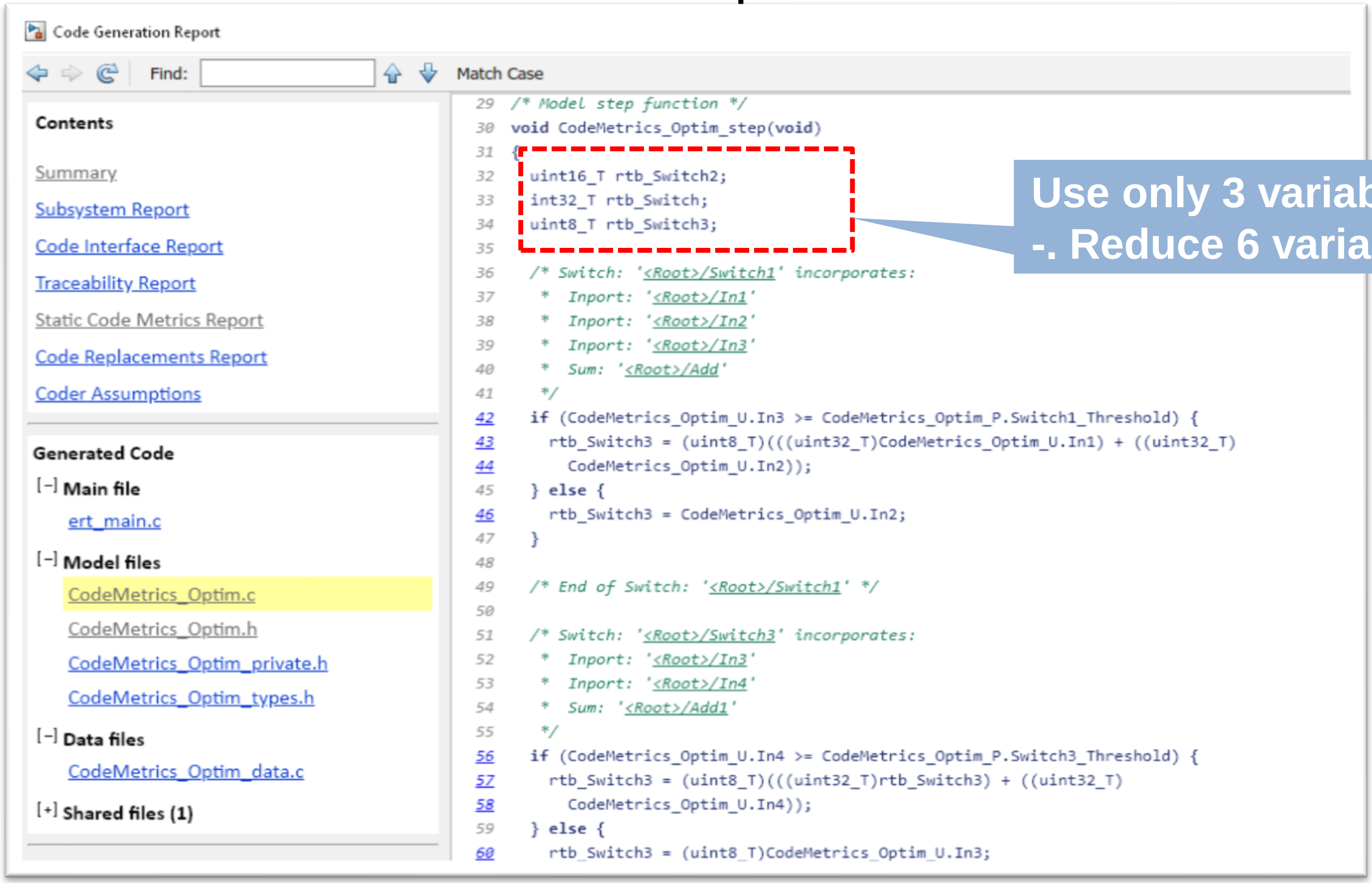
MATLAB EXPO

MathWorks

Reuse Local Block Output

RAM Efficiency

- Review Code Generation Report



Contents

Summary

Subsystem Report

Code Interface Report

Traceability Report

Static Code Metrics Report

Code Replacements Report

Coder Assumptions

Generated Code

[-] **Main file**

ert_main.c

[-] **Model files**

CodeMetrics_Optim.c

CodeMetrics_Optim.h

CodeMetrics_Optim_private.h

CodeMetrics_Optim_types.h

[-] **Data files**

CodeMetrics_Optim_data.c

[+] **Shared files (1)**

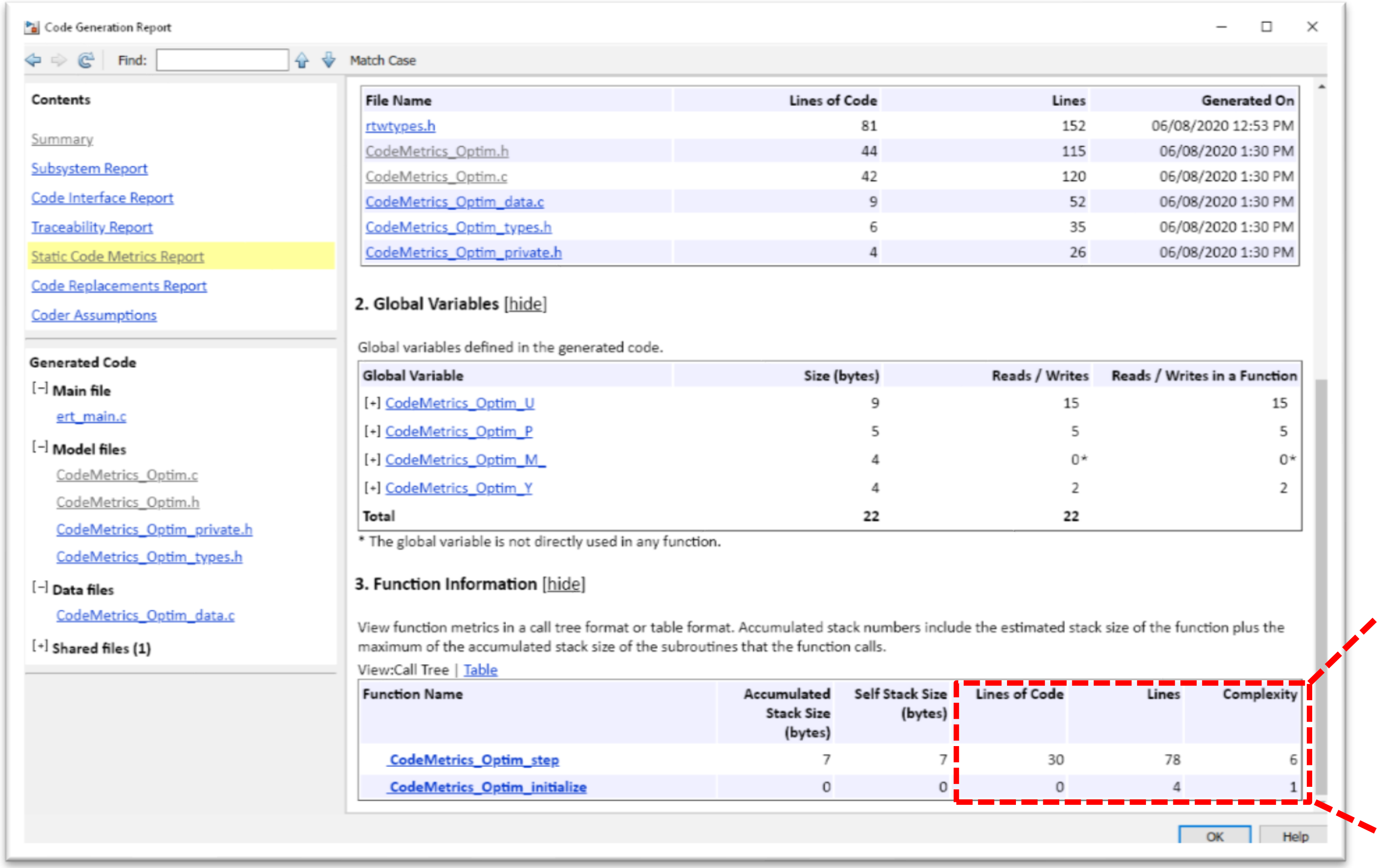
```c
/* Model step function */
void CodeMetrics_Optim_step(void)
{
  uint16_T rtb_Switch2;
  int32_T rtb_Switch;
  uint8_T rtb_Switch3;

  /* Switch: '<Root>/Switch1' incorporates:
   *  Inport: '<Root>/In1'
   *  Inport: '<Root>/In2'
   *  Inport: '<Root>/In3'
   *  Sum: '<Root>/Add'
   */
  if (CodeMetrics_Optim_U.In3 >= CodeMetrics_Optim_P.Switch1_Threshold) {
    rtb_Switch3 = (uint8_T)(((uint32_T)CodeMetrics_Optim_U.In1) + ((uint32_T)
      CodeMetrics_Optim_U.In2));
  } else {
    rtb_Switch3 = CodeMetrics_Optim_U.In2;
  }

  /* End of Switch: '<Root>/Switch1' */

  /* Switch: '<Root>/Switch3' incorporates:
   *  Inport: '<Root>/In3'
   *  Inport: '<Root>/In4'
   *  Sum: '<Root>/Add1'
   */
  if (CodeMetrics_Optim_U.In4 >= CodeMetrics_Optim_P.Switch3_Threshold) {
    rtb_Switch3 = (uint8_T)(((uint32_T)rtb_Switch3) + ((uint32_T)
      CodeMetrics_Optim_U.In4));
  } else {
    rtb_Switch3 = (uint8_T)CodeMetrics_Optim_U.In3;
```

Use only 3 variables for 9 blocks
-. Reduce 6 variables

Reuse Local Block Output

RAM Efficiency

- Review Static Code Metrics Report



Contents

Summary
Subsystem Report
Code Interface Report
Traceability Report
Static Code Metrics Report
Code Replacements Report
Coder Assumptions

Generated Code

[-] Main file
ert_main.c

[-] Model files
CodeMetrics_Optim.c
CodeMetrics_Optim.h
CodeMetrics_Optim_private.h
CodeMetrics_Optim_types.h

[-] Data files
CodeMetrics_Optim_data.c

[+] Shared files (1)

File Name	Lines of Code	Lines	Generated On
rtwtypes.h	81	152	06/08/2020 12:53 PM
CodeMetrics_Optim.h	44	115	06/08/2020 1:30 PM
CodeMetrics_Optim.c	42	120	06/08/2020 1:30 PM
CodeMetrics_Optim_data.c	9	52	06/08/2020 1:30 PM
CodeMetrics_Optim_types.h	6	35	06/08/2020 1:30 PM
CodeMetrics_Optim_private.h	4	26	06/08/2020 1:30 PM

2. Global Variables [hide]

Global variables defined in the generated code.

Global Variable	Size (bytes)	Reads / Writes	Reads / Writes in a Function
[+] CodeMetrics_Optim_U	9	15	15
[+] CodeMetrics_Optim_P	5	5	5
[+] CodeMetrics_Optim_M_	4	0*	0*
[+] CodeMetrics_Optim_Y	4	2	2
Total	22	22	

* The global variable is not directly used in any function.

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.

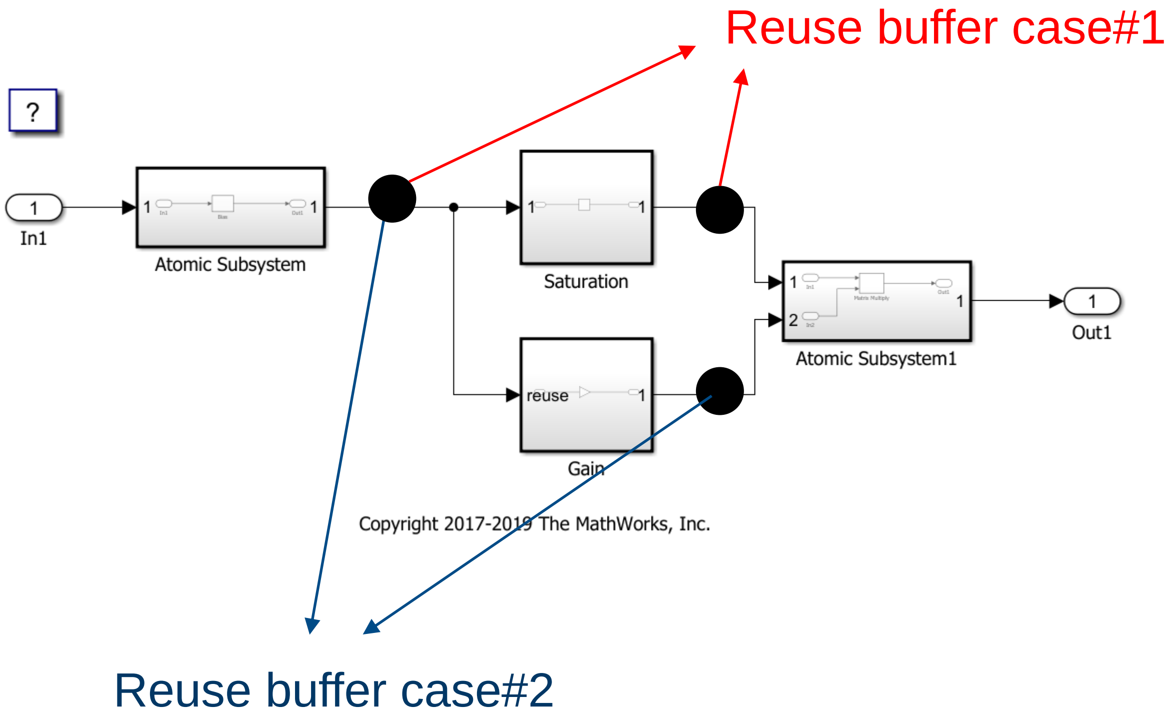
View:Call Tree | Table

Function Name	Accumulated Stack Size (bytes)	Self Stack Size (bytes)	Lines of Code	Lines	Complexity
CodeMetrics_Optim_step	7	7	30	78	6
CodeMetrics_Optim_initialize	0	0	0	4	1

	Before	After
Lines of Code	33	30
Total Lines	89	78

Reuse Buffer Using Signal Labels

- Using Signal Labels to Guide Buffer Reuse
  - Case#1: Same variable for the Atomic Subsystem and Saturation block outputs
  - Case#2: Same variable for the Atomic Subsystem and Gain block outputs



Reuse buffer case#1

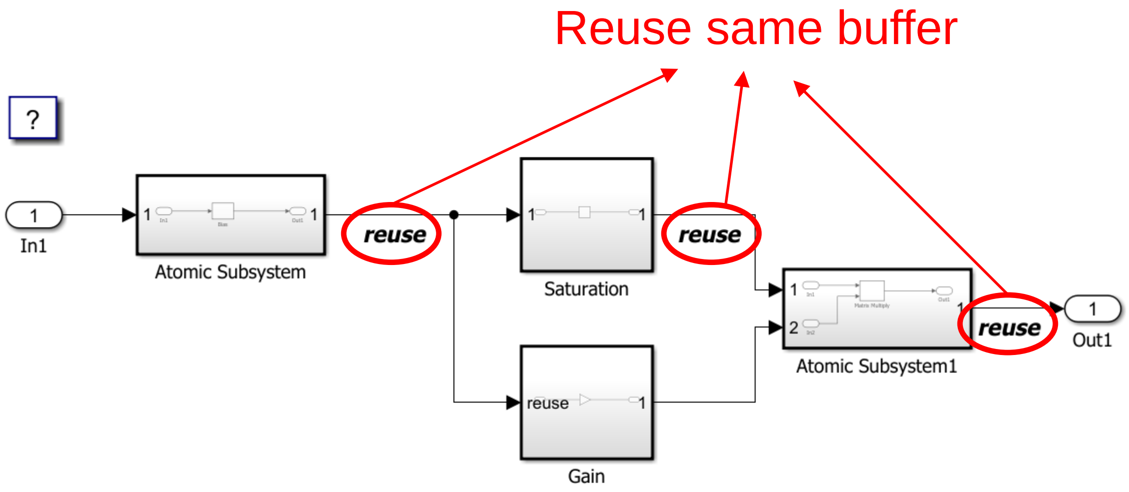
Reuse buffer case#2

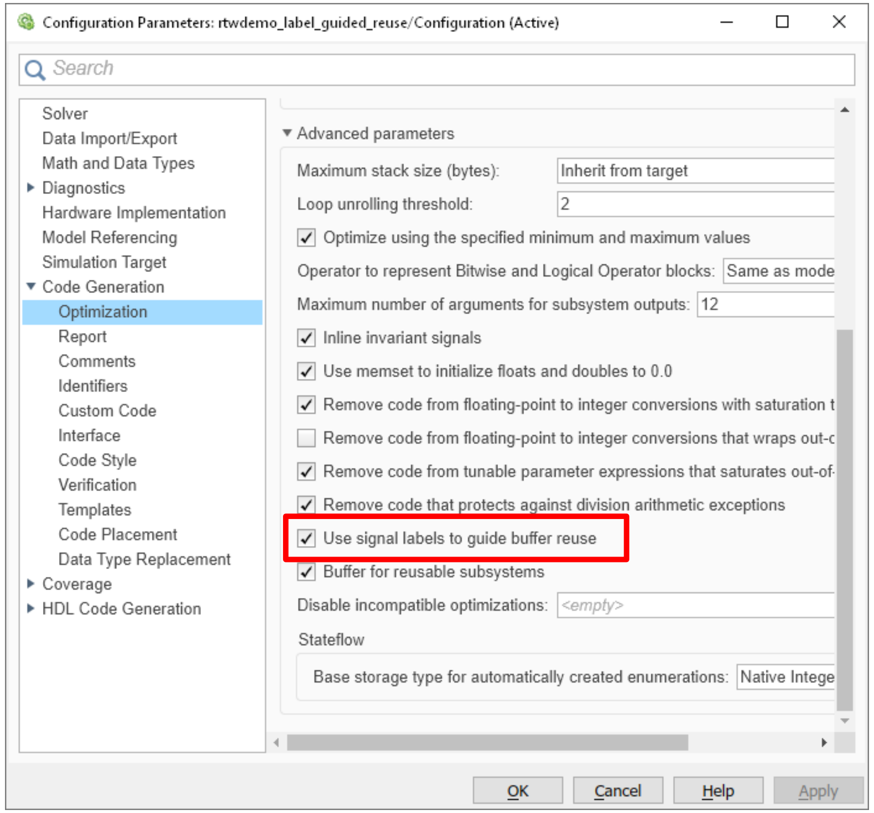
Reuse Buffer Using Signal Labels

RAM Efficiency

- Using Signal Labels to Guide Buffer Reuse
  - Same variable for the Atomic Subsystem and Saturation block outputs

Reuse same buffer





Configuration Parameters: rtwdemo_label_guided_reuse/Configuration (Active)

▾ Advanced parameters

Maximum stack size (bytes): Inherit from target

Loop unrolling threshold: 2

☑ Optimize using the specified minimum and maximum values

Operator to represent Bitwise and Logical Operator blocks: Same as mode

Maximum number of arguments for subsystem outputs: 12

☑ Inline invariant signals

☑ Use memset to initialize floats and doubles to 0.0

☑ Remove code from floating-point to integer conversions with saturation t

□ Remove code from floating-point to integer conversions that wraps out-c

☑ Remove code from tunable parameter expressions that saturates out-of-

☑ Remove code that protects against division arithmetic exceptions

☑ Use signal labels to guide buffer reuse

☑ Buffer for reusable subsystems

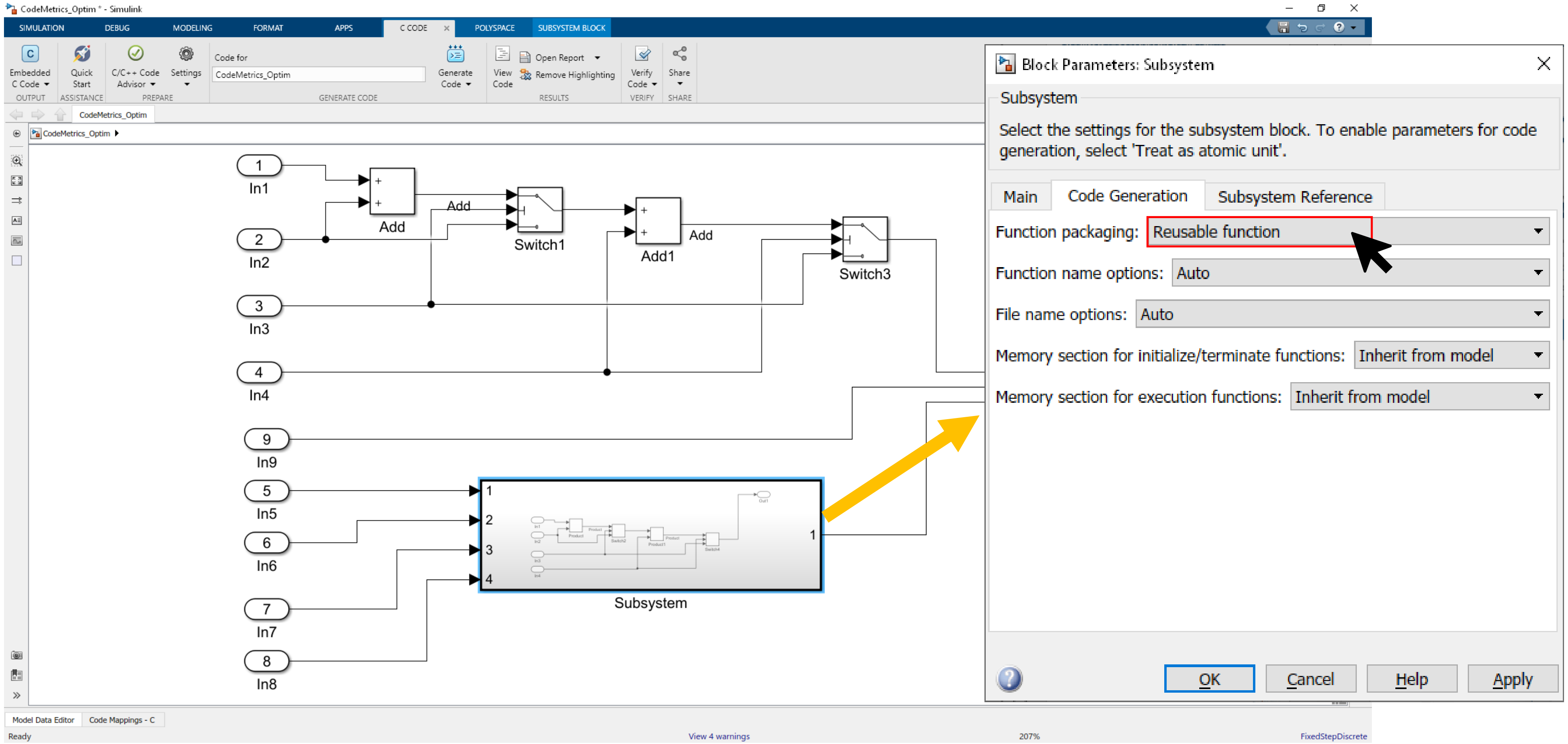
Disable incompatible optimizations: <empty>

Stateflow

Base storage type for automatically created enumerations: Native Intege

Reduce Code Complexity by Refactoring Subsystem

RAM Efficiency



Block Parameters: Subsystem

Subsystem

Select the settings for the subsystem block. To enable parameters for code generation, select 'Treat as atomic unit'.

Function packaging: Reusable function

Function name options: Auto

File name options: Auto

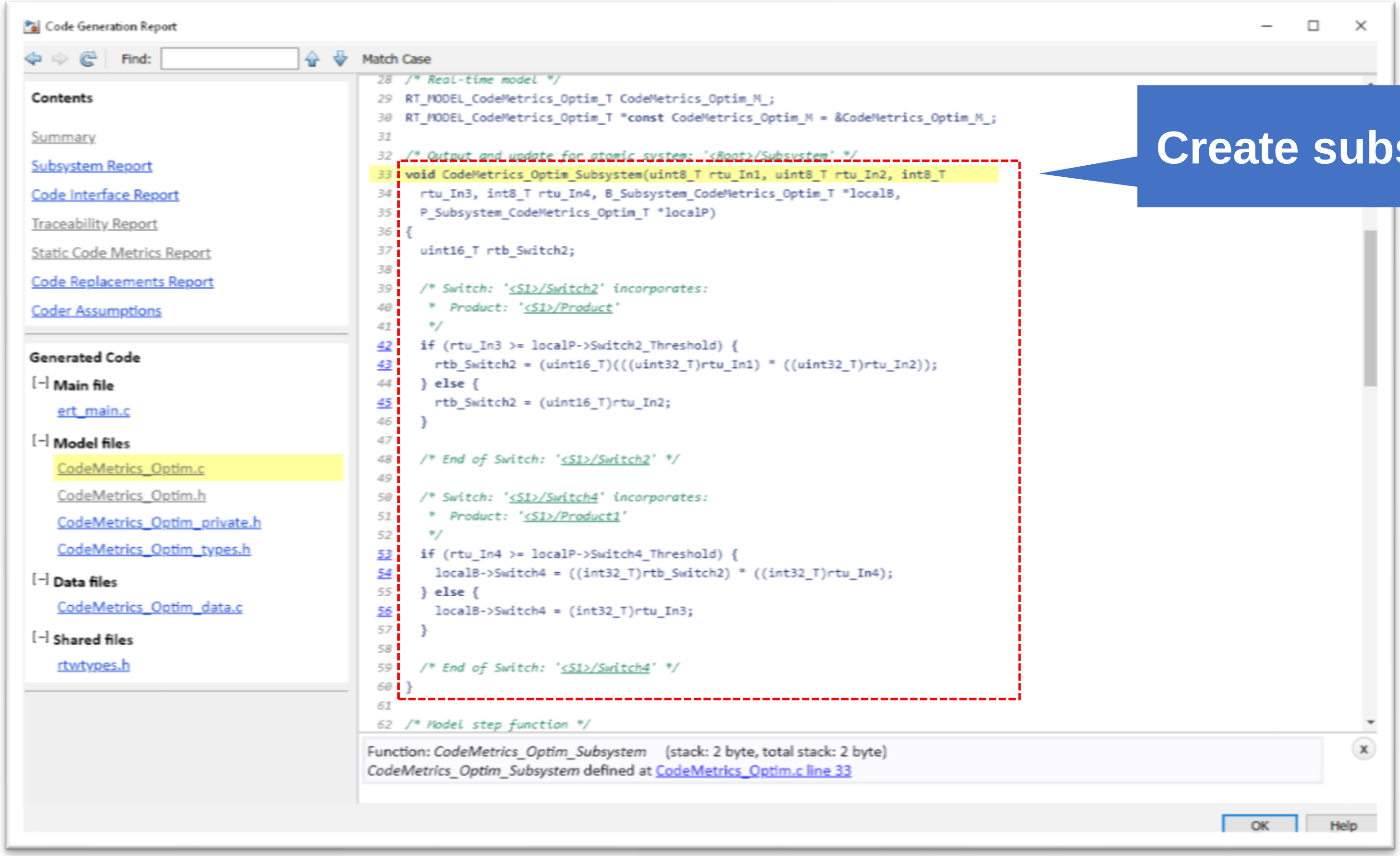
Memory section for initialize/terminate functions: Inherit from model

Memory section for execution functions: Inherit from model

Reduce Code Complexity by Refactoring Subsystem

RAM Efficiency

- Review Code Generation Report



```
/* Real-time model */
RT_MODEL_CodeMetrics_Optim_T CodeMetrics_Optim_M_;
RT_MODEL_CodeMetrics_Optim_T *const CodeMetrics_Optim_M = &CodeMetrics_Optim_M_;

/* Output and update for atomic system: '<Root>/Subsystem' */
void CodeMetrics_Optim_Subsystem(uint8_T rtu_In1, uint8_T rtu_In2, int8_T
  rtu_In3, int8_T rtu_In4, B_Subsystem_CodeMetrics_Optim_T *localB,
  P_Subsystem_CodeMetrics_Optim_T *localP)
{
  uint16_T rtb_Switch2;

  /* Switch: '<S1>/Switch2' incorporates:
   *  Product: '<S1>/Product'
   */
  if (rtu_In3 >= localP->Switch2_Threshold) {
    rtb_Switch2 = (uint16_T)(((uint32_T)rtu_In1) * ((uint32_T)rtu_In2));
  } else {
    rtb_Switch2 = (uint16_T)rtu_In2;
  }

  /* End of Switch: '<S1>/Switch2' */

  /* Switch: '<S1>/Switch4' incorporates:
   *  Product: '<S1>/Product1'
   */
  if (rtu_In4 >= localP->Switch4_Threshold) {
    localB->Switch4 = ((int32_T)rtb_Switch2) * ((int32_T)rtu_In4);
  } else {
    localB->Switch4 = (int32_T)rtu_In3;
  }

  /* End of Switch: '<S1>/Switch4' */
}

/* Model step function */
```

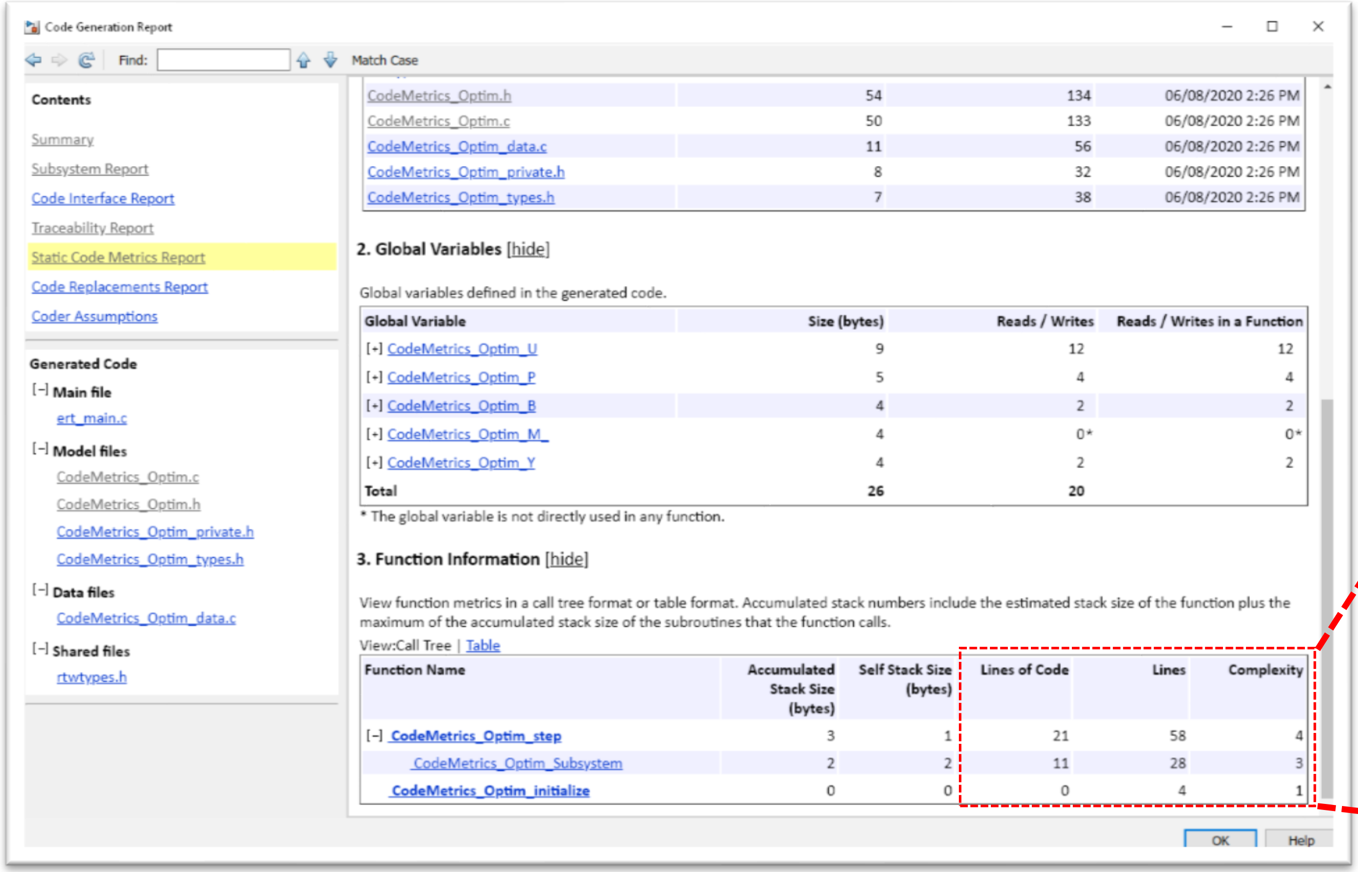
Function: *CodeMetrics_Optim_Subsystem* (stack: 2 byte, total stack: 2 byte)
CodeMetrics_Optim_Subsystem defined at CodeMetrics_Optim.c line 33

Create subsystem function

Reduce Code Complexity by Refactoring Subsystem

RAM Efficiency

- Review Static Code Metrics Report



CodeMetrics_Optim.h	54	134	06/08/2020 2:26 PM
CodeMetrics_Optim.c	50	133	06/08/2020 2:26 PM
CodeMetrics_Optim_data.c	11	56	06/08/2020 2:26 PM
CodeMetrics_Optim_private.h	8	32	06/08/2020 2:26 PM
CodeMetrics_Optim_types.h	7	38	06/08/2020 2:26 PM

2. Global Variables [hide]

Global variables defined in the generated code.

Global Variable	Size (bytes)	Reads / Writes	Reads / Writes in a Function
[+] CodeMetrics_Optim_U	9	12	12
[+] CodeMetrics_Optim_P	5	4	4
[+] CodeMetrics_Optim_B	4	2	2
[+] CodeMetrics_Optim_M_	4	0*	0*
[+] CodeMetrics_Optim_Y	4	2	2
Total	**26**	**20**	

* The global variable is not directly used in any function.

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

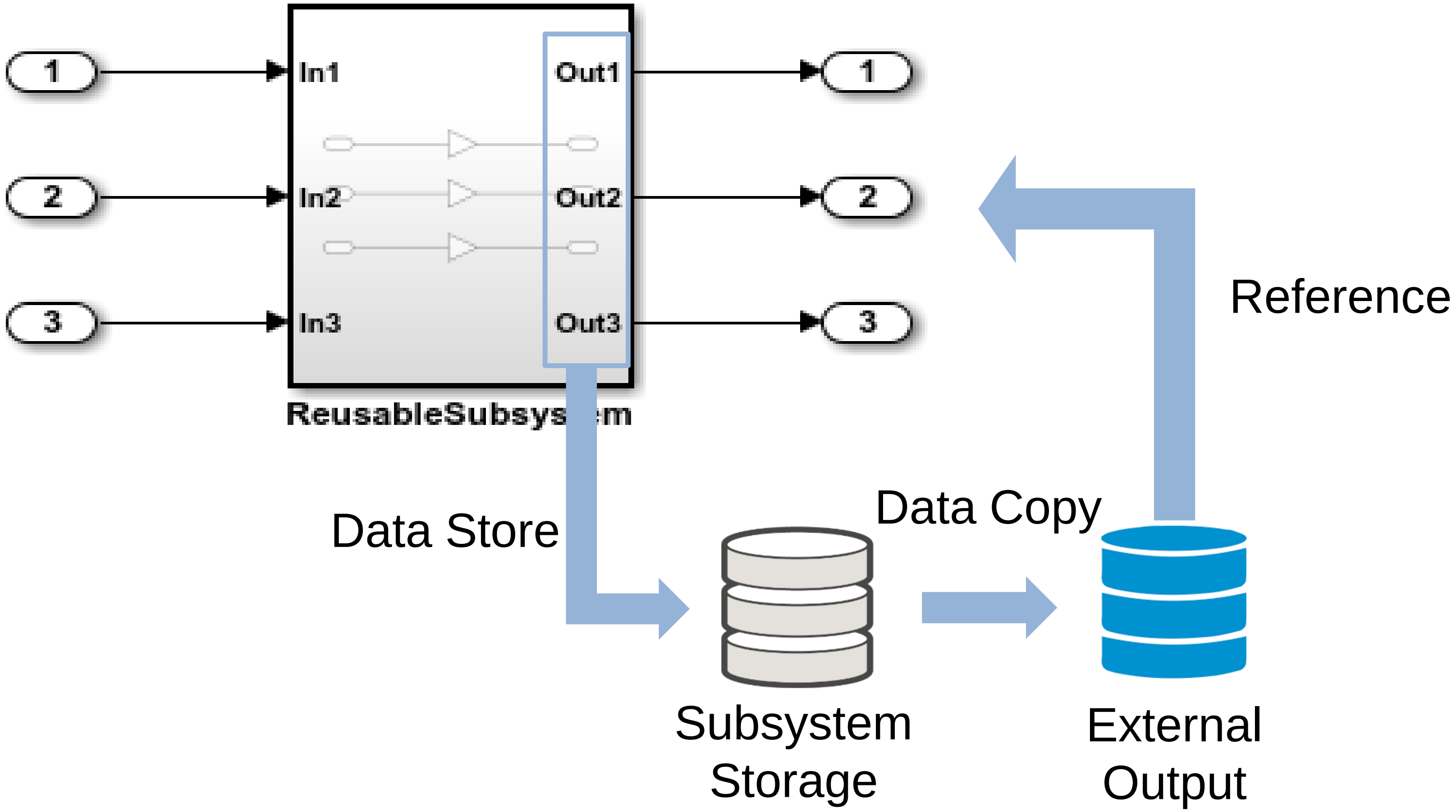
Function Name	Accumulated Stack Size (bytes)	Self Stack Size (bytes)	Lines of Code	Lines	Complexity
[-] CodeMetrics_Optim_step	3	1	21	58	4
CodeMetrics_Optim_Subsystem	2	2	11	28	3
CodeMetrics_Optim_initialize	0	0	0	4	1

	Before	After
Complexity	6	4
Lines of Code	33	11
Total Lines	78	58

Optimize Generate Code in Reusable Subsystem

RAM Efficiency

- Passing Reusable Subsystem Outputs as *Structure Reference*



Reference

Data Store

Data Copy

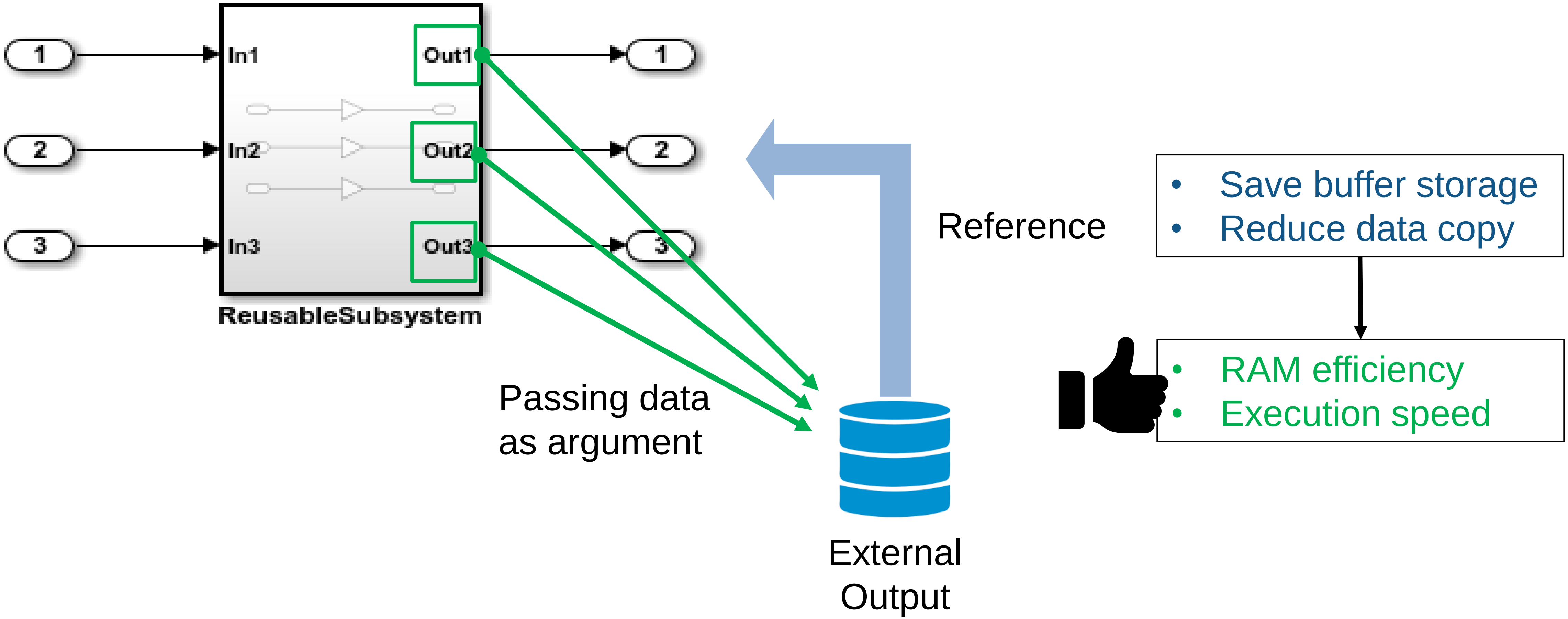
Subsystem Storage

External Output

Optimize Generate Code in Reusable Subsystem

RAM Efficiency

- Passing Reusable Subsystem Outputs as *Individual Argument*



Reference

Passing data as argument

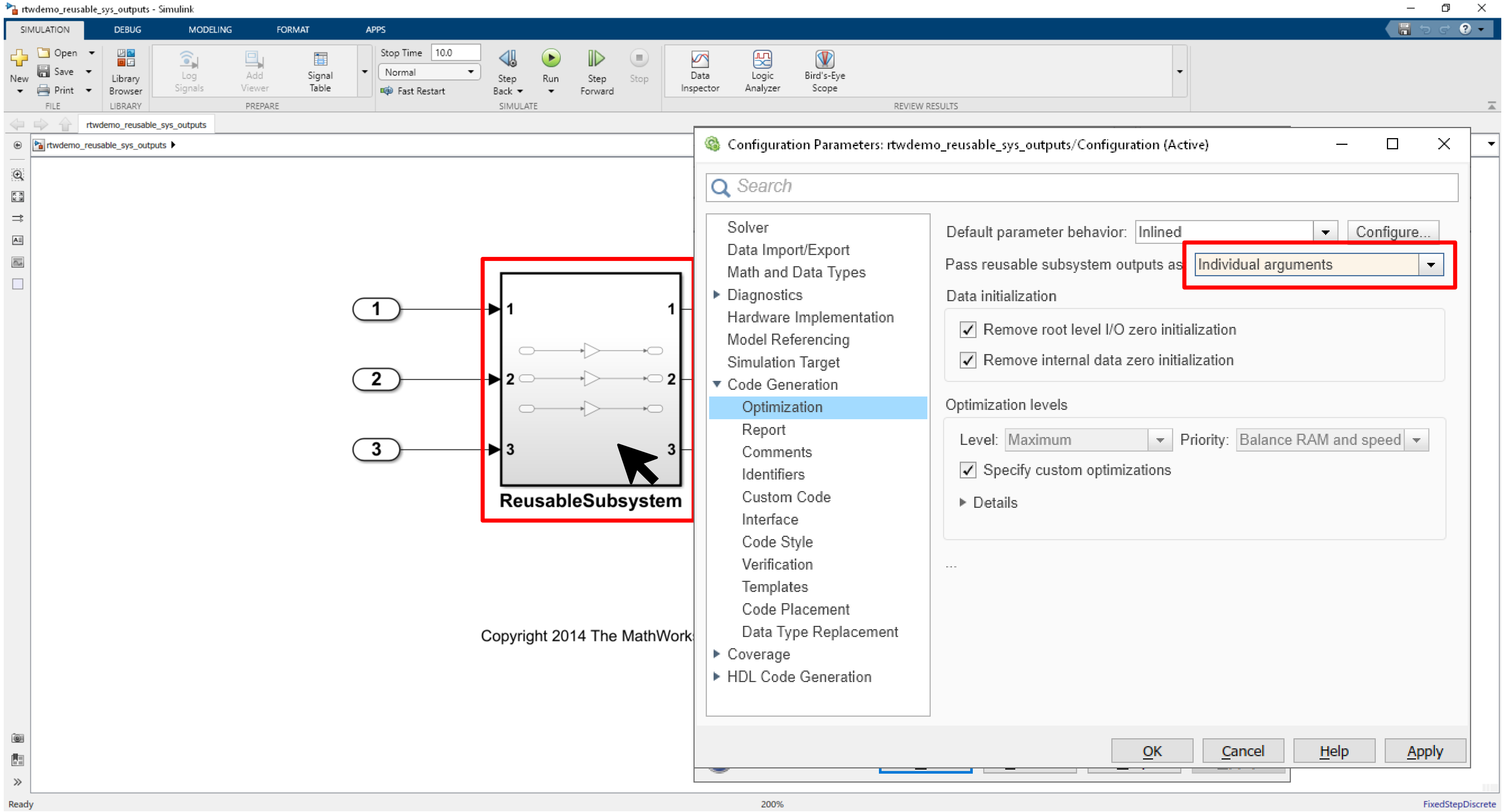
External Output

- Save buffer storage
- Reduce data copy

- RAM efficiency
- Execution speed

Optimize Generate Code in Reusable Subsystem

RAM Efficiency



Default parameter behavior: Inlined

Pass reusable subsystem outputs as: Individual arguments

Data initialization

☑ Remove root level I/O zero initialization

☑ Remove internal data zero initialization

Optimization levels

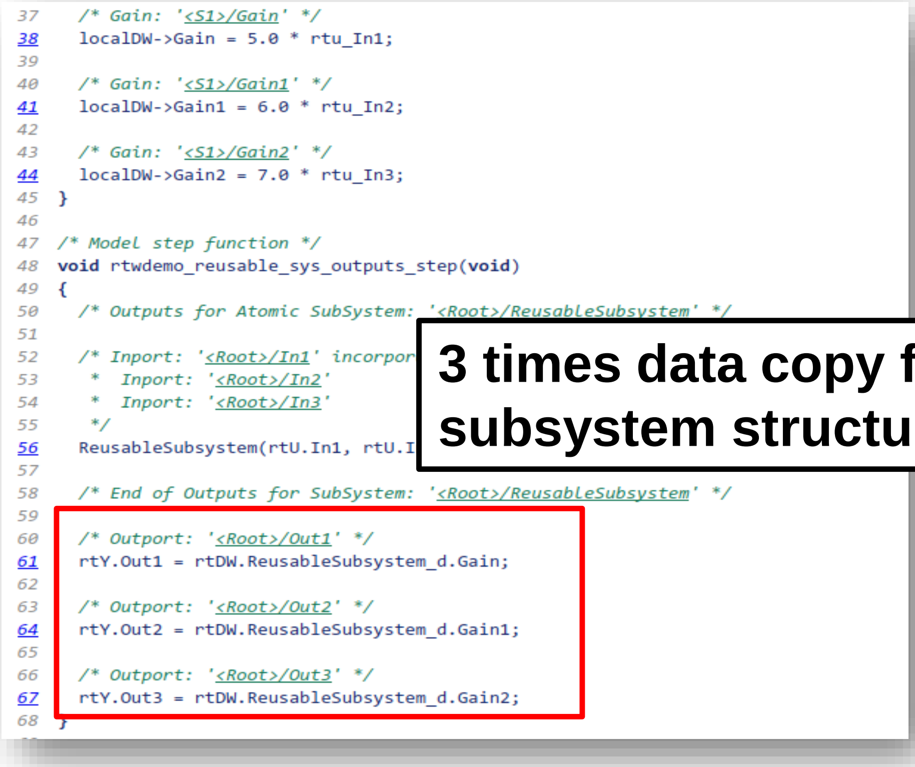
Level: Maximum Priority: Balance RAM and speed

☑ Specify custom optimizations

Optimize Generate Code in Reusable Subsystem

RAM Efficiency

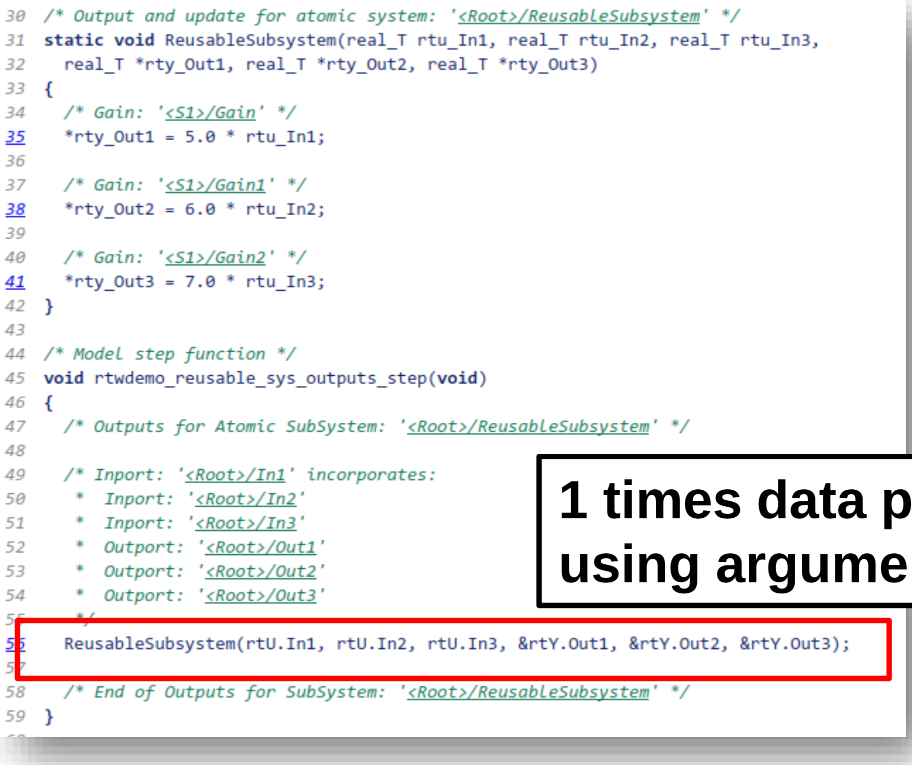
- Review Code Generation Report



```
  /* Gain: '<S1>/Gain' */
  localDW->Gain = 5.0 * rtu_In1;

  /* Gain: '<S1>/Gain1' */
  localDW->Gain1 = 6.0 * rtu_In2;

  /* Gain: '<S1>/Gain2' */
  localDW->Gain2 = 7.0 * rtu_In3;
}

/* Model step function */
void rtwdemo_reusable_sys_outputs_step(void)
{
  /* Outputs for Atomic SubSystem: '<Root>/ReusableSubsystem' */

  /* Inport: '<Root>/In1' incorpor
   *  Inport: '<Root>/In2'
   *  Inport: '<Root>/In3'
   */
  ReusableSubsystem(rtU.In1, rtU.I

  /* End of Outputs for SubSystem: '<Root>/ReusableSubsystem' */

  /* Outport: '<Root>/Out1' */
  rtY.Out1 = rtDW.ReusableSubsystem_d.Gain;

  /* Outport: '<Root>/Out2' */
  rtY.Out2 = rtDW.ReusableSubsystem_d.Gain1;

  /* Outport: '<Root>/Out3' */
  rtY.Out3 = rtDW.ReusableSubsystem_d.Gain2;
}
```

3 times data copy from subsystem structure

[Structure reference]



```
/* Output and update for atomic system: '<Root>/ReusableSubsystem' */
static void ReusableSubsystem(real_T rtu_In1, real_T rtu_In2, real_T rtu_In3,
  real_T *rty_Out1, real_T *rty_Out2, real_T *rty_Out3)
{
  /* Gain: '<S1>/Gain' */
  *rty_Out1 = 5.0 * rtu_In1;

  /* Gain: '<S1>/Gain1' */
  *rty_Out2 = 6.0 * rtu_In2;

  /* Gain: '<S1>/Gain2' */
  *rty_Out3 = 7.0 * rtu_In3;
}

/* Model step function */
void rtwdemo_reusable_sys_outputs_step(void)
{
  /* Outputs for Atomic SubSystem: '<Root>/ReusableSubsystem' */

  /* Inport: '<Root>/In1' incorporates:
   *  Inport: '<Root>/In2'
   *  Inport: '<Root>/In3'
   *  Outport: '<Root>/Out1'
   *  Outport: '<Root>/Out2'
   *  Outport: '<Root>/Out3'
   */
  ReusableSubsystem(rtU.In1, rtU.In2, rtU.In3, &rtY.Out1, &rtY.Out2, &rtY.Out3);

  /* End of Outputs for SubSystem: '<Root>/ReusableSubsystem' */
}
```

1 times data passing using arguments

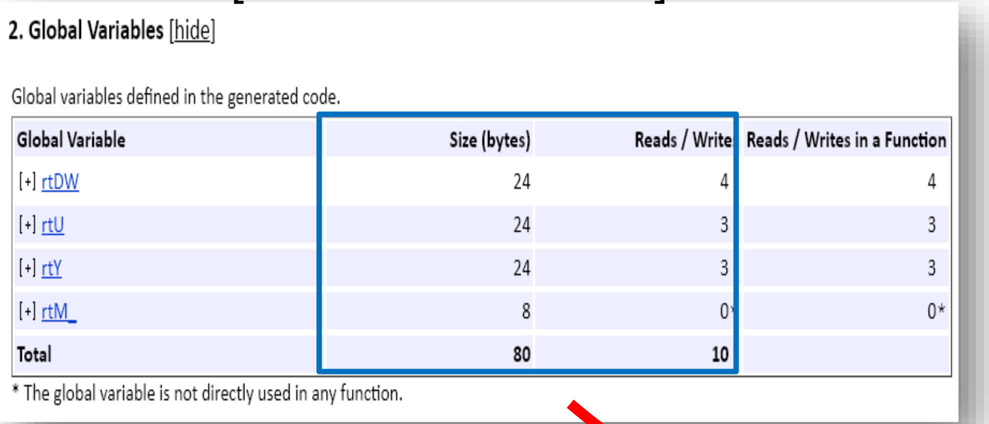
[Individual argument]

Optimize Generate Code in Reusable Subsystem

RAM Efficiency

- Review Static Code Metrics Report

[Structure reference]



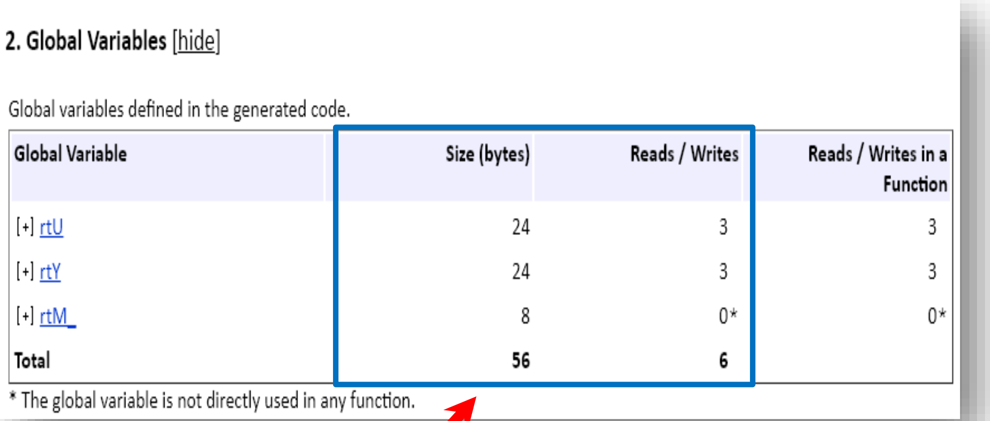
2. Global Variables [hide]

Global variables defined in the generated code.

Global Variable	Size (bytes)	Reads / Write	Reads / Writes in a Function
[+] rtDW	24	4	4
[+] rtU	24	3	3
[+] rtY	24	3	3
[+] rtM_	8	0*	0*
Total	**80**	**10**	

* The global variable is not directly used in any function.

[Individual argument]



2. Global Variables [hide]

Global variables defined in the generated code.

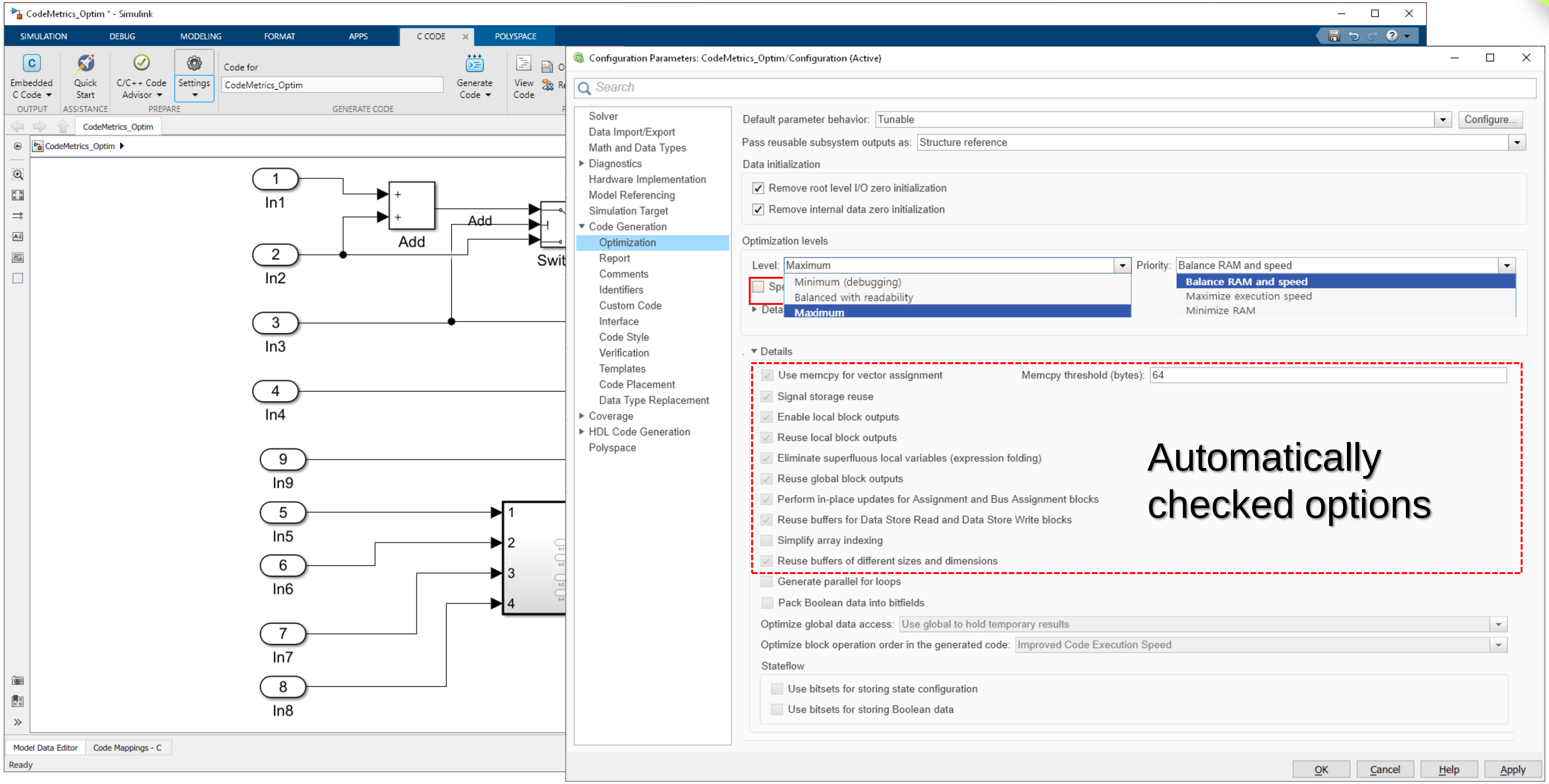
Global Variable	Size (bytes)	Reads / Writes	Reads / Writes in a Function
[+] rtU	24	3	3
[+] rtY	24	3	3
[+] rtM_	8	0*	0*
Total	**56**	**6**	

* The global variable is not directly used in any function.

Global variables: -24bytes
Read/Write: -4 times

Easy to Configure Options for Optimizing Code

RAM Efficiency



Default parameter behavior: Tunable

Pass reusable subsystem outputs as: Structure reference

Data initialization

☑ Remove root level I/O zero initialization

☑ Remove internal data zero initialization

Optimization levels

Level: Maximum
- Minimum (debugging)
- Balanced with readability
- Maximum

Priority: Balance RAM and speed
- Balance RAM and speed
- Maximize execution speed
- Minimize RAM

☐ Spe

Deta

Details

☑ Use memcpy for vector assignment — Memcpy threshold (bytes): 64

☑ Signal storage reuse

☑ Enable local block outputs

☑ Reuse local block outputs

☑ Eliminate superfluous local variables (expression folding)

☑ Reuse global block outputs

☑ Perform in-place updates for Assignment and Bus Assignment blocks

☑ Reuse buffers for Data Store Read and Data Store Write blocks

☐ Simplify array indexing

☑ Reuse buffers of different sizes and dimensions

☐ Generate parallel for loops

☐ Pack Boolean data into bitfields

Optimize global data access: Use global to hold temporary results

Optimize block operation order in the generated code: Improved Code Execution Speed

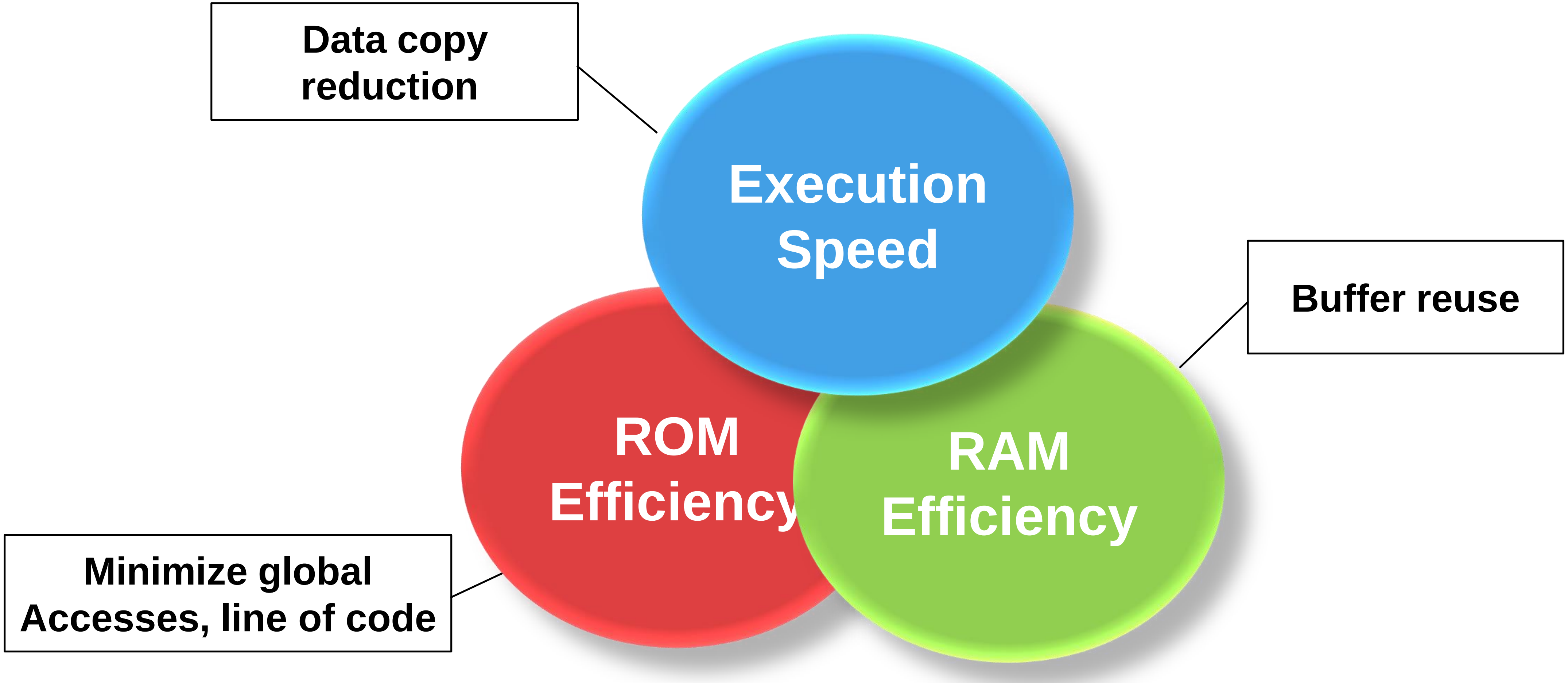
Stateflow

☐ Use bitsets for storing state configuration

☐ Use bitsets for storing Boolean data

Automatically checked options

RAM, ROM and Execution Performance



Execution Speed

ROM Efficiency

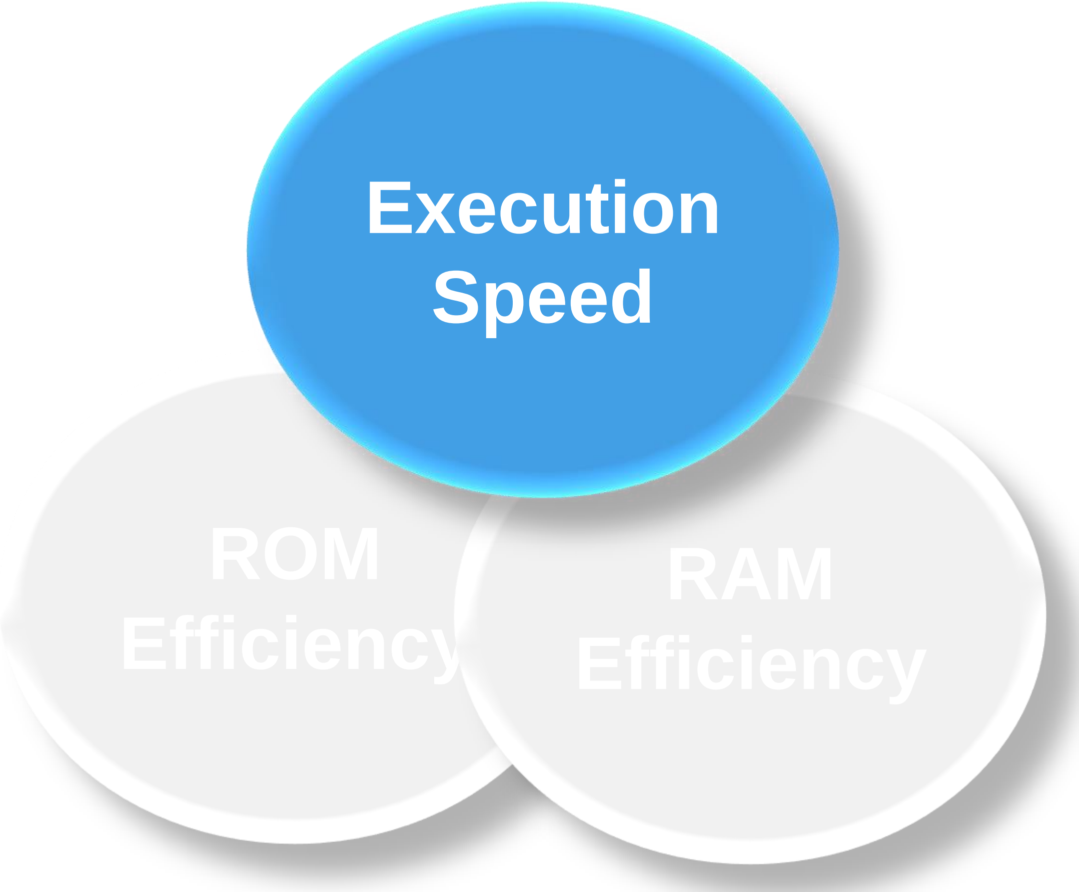
RAM Efficiency

Data copy reduction

Buffer reuse

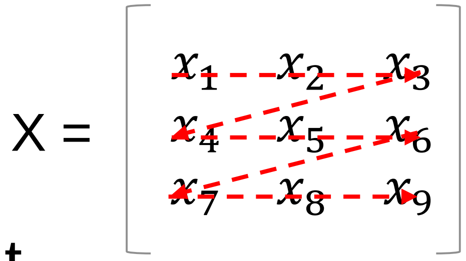
Minimize global Accesses, line of code

RAM, ROM and Execution Performance



Row-Major vs. Column-Major

- Row-Major layout
  - Elements of the rows are contiguous
  - C and C++ use row-major layout

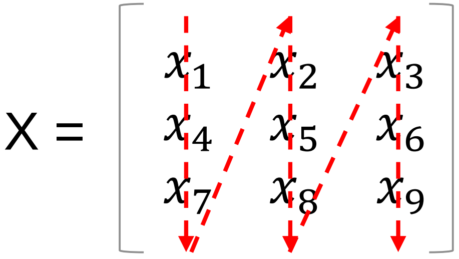


$$X = \begin{bmatrix} x_1 & x_2 & x_3 \\ x_4 & x_5 & x_6 \\ x_7 & x_8 & x_9 \end{bmatrix}$$

The elements of the array are stored :

$x_1\ x_2\ x_3\ x_4\ x_5\ x_6\ x_7\ x_8\ x_9$

- Column-Major layout
  - Elements of the columns are contiguous
  - MATLAB® and Fortran use column-major layout



$$X = \begin{bmatrix} x_1 & x_2 & x_3 \\ x_4 & x_5 & x_6 \\ x_7 & x_8 & x_9 \end{bmatrix}$$

The elements of the array are stored :

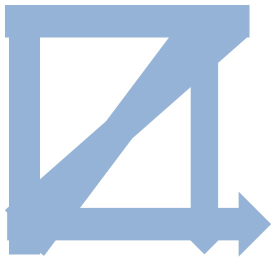
$x_1\ x_4\ x_7\ x_2\ x_5\ x_8\ x_3\ x_6\ x_9$

Row-Major vs. Column-Major

Execution Speed

Column-major code generation: M[] = {11, 21, 31, 12, 22, 32}; → M[2] = 31 Column-major indexing

MATLAB: $M = \begin{bmatrix} 11 & 12 \\ 21 & 22 \\ 31 & 32 \end{bmatrix}$



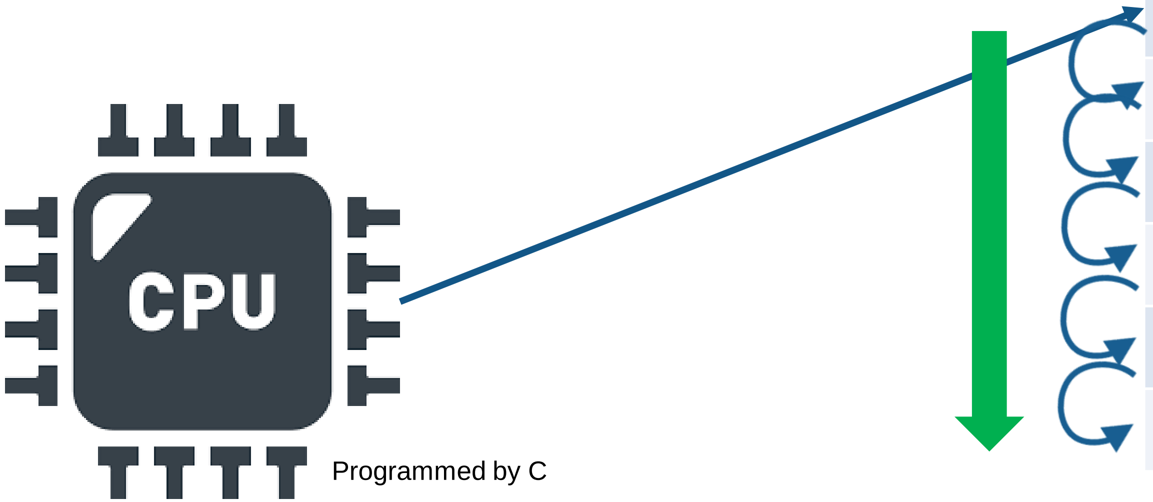
Row-major code generation: M[] = {11, 12, 21, 22, 31, 32}; → M[2] = **21** Row-major indexing

Row-Major vs. Column-Major

Execution Speed

- CPUs Process Sequential Data More Efficiently than nonsequential data

$$A = \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \end{bmatrix}$$



$a_{11}, a_{12}, a_{13}, \ldots\ldots.$

Sequentially data read

Address	Access	Value
0	A[0][0]	a_{11}
1	A[0][1]	a_{12}
2	A[0][2]	a_{13}
3	A[1][0]	a_{21}
4	A[1][1]	a_{22}
5	A[1][2]	a_{23}

[Raw-major order]

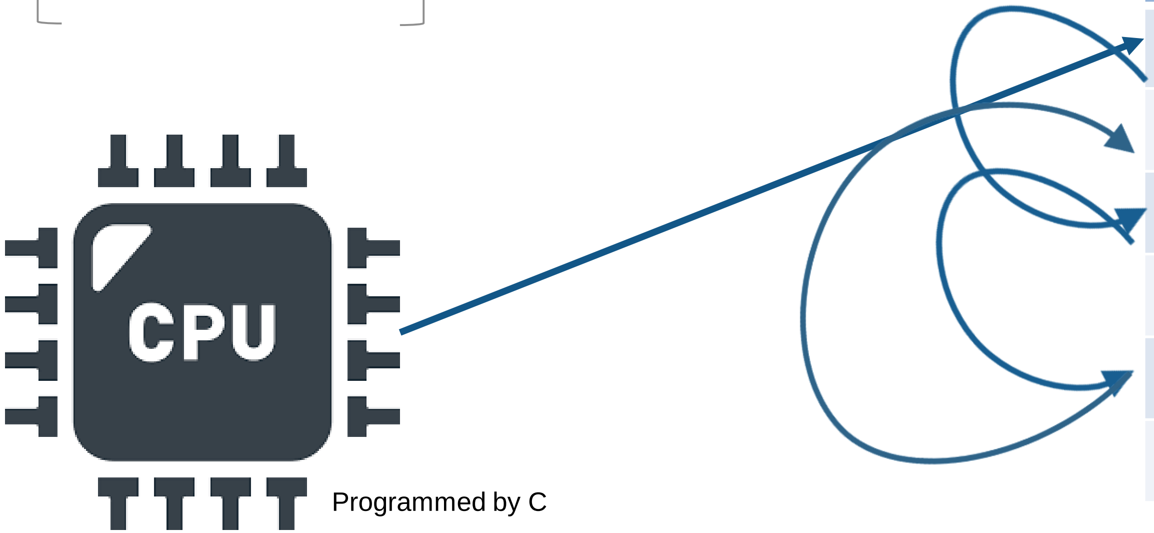
Row-Major vs. Column-Major

Execution Speed

- CPUs Process Sequential Data More Efficiently than nonsequential data

$$A = \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \end{bmatrix}$$

Need data indexing!!



Address	Access	Value
0	A[0][0]	a_{11}
1	A[1][0]	a_{21}
2	A[0][1]	a_{12}
3	A[1][1]	a_{22}
4	A[0][2]	a_{13}
5	A[1][2]	a_{23}

[Column major order]

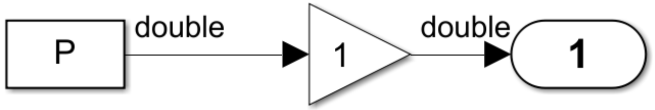
$a_{11}, a_{12}, a_{13}, \ldots\ldots.$

Memory access times increase!!

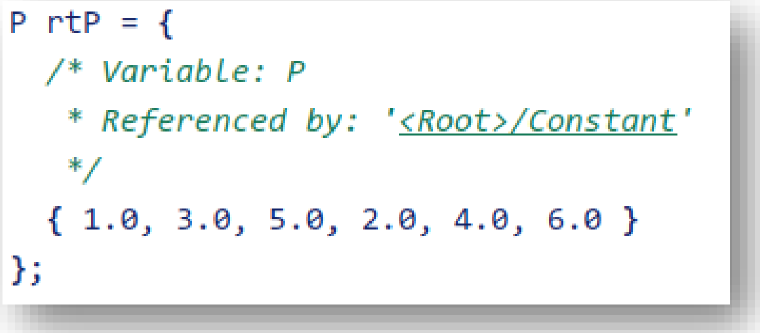
Row-Major vs. Column-Major

Execution Speed

$$P = \begin{bmatrix} 1 & 2 \\ 3 & 4 \\ 5 & 6 \end{bmatrix}$$

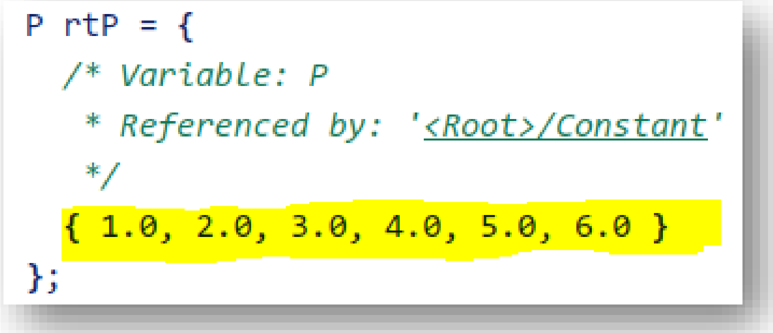


Column-major layout



```
P rtP = {
  /* Variable: P
   * Referenced by: '<Root>/Constant'
   */
  { 1.0, 3.0, 5.0, 2.0, 4.0, 6.0 }
};
```

Row-major layout

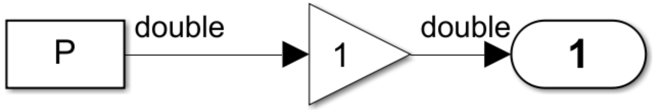


```
P rtP = {
  /* Variable: P
   * Referenced by: '<Root>/Constant'
   */
  { 1.0, 2.0, 3.0, 4.0, 5.0, 6.0 }
};
```

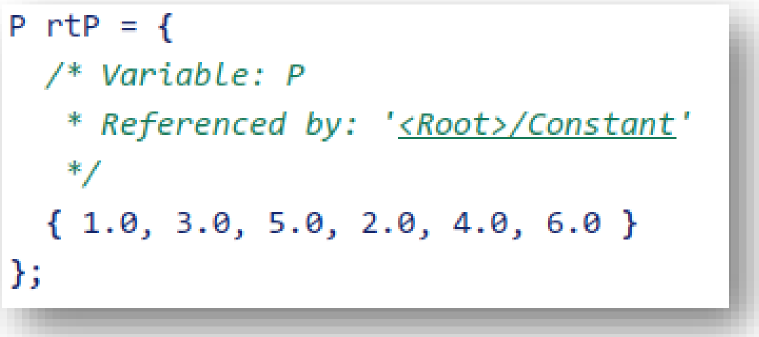
Row-Major and Multi-Dimension Indexing

Execution Speed

$$P = \begin{bmatrix} 1 & 2 \\ 3 & 4 \\ 5 & 6 \end{bmatrix}$$

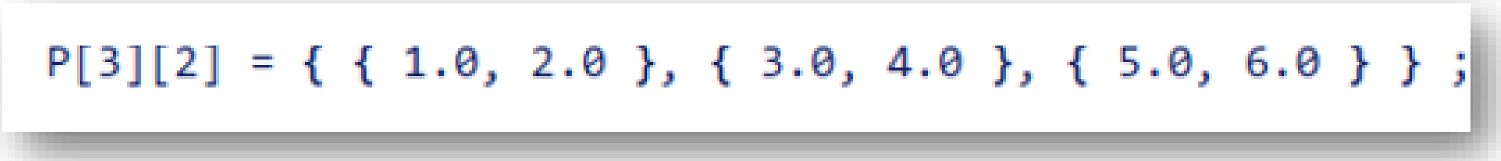


Row-major layout



```
P rtP = {
  /* Variable: P
   * Referenced by: '<Root>/Constant'
   */
  { 1.0, 3.0, 5.0, 2.0, 4.0, 6.0 }
};
```

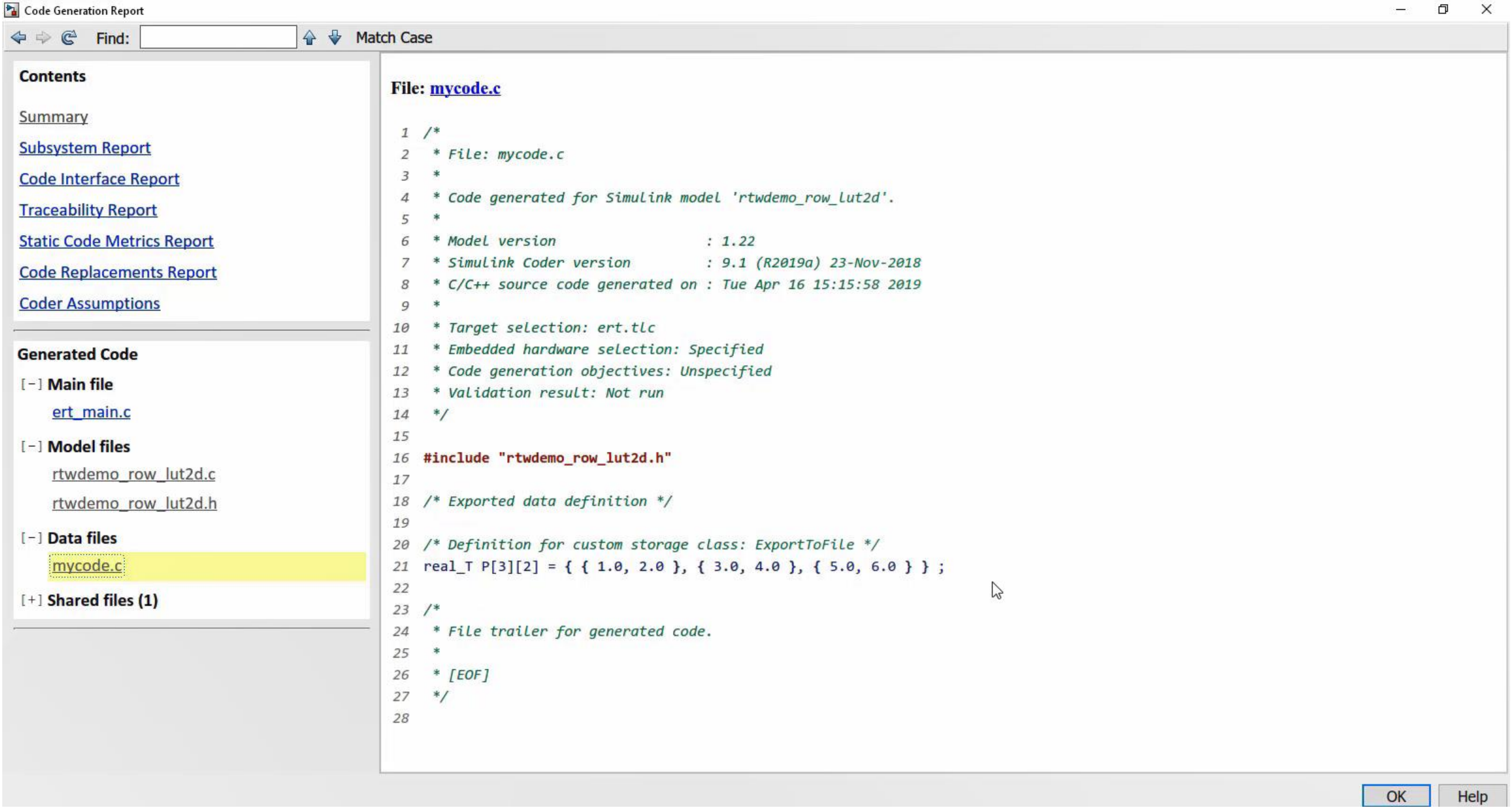
Multi-Dimensional layout



```
P[3][2] = { { 1.0, 2.0 }, { 3.0, 4.0 }, { 5.0, 6.0 } } ;
```

Generating Row-Major Code

Execution Speed

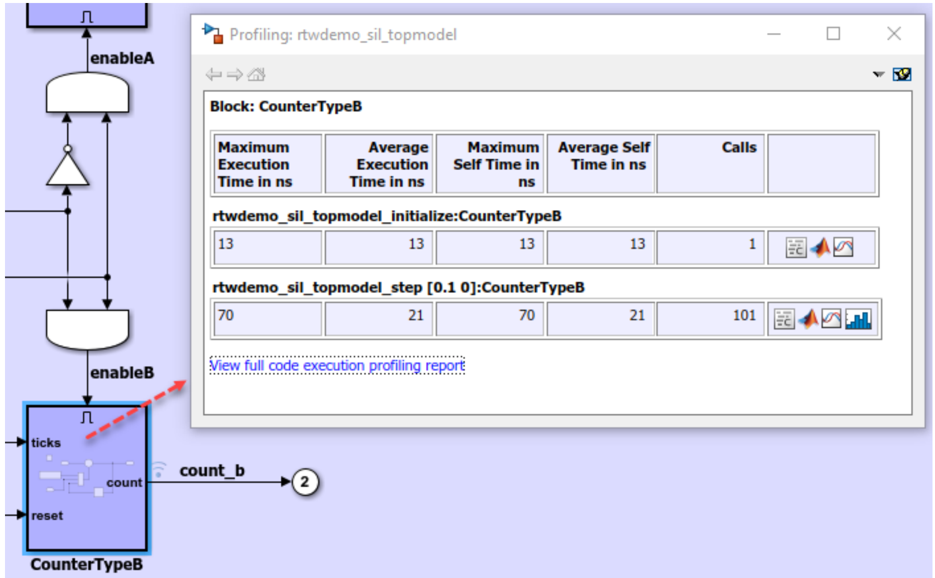


File: mycode.c

```
/*
 * File: mycode.c
 *
 * Code generated for Simulink model 'rtwdemo_row_lut2d'.
 *
 * Model version                   : 1.22
 * Simulink Coder version          : 9.1 (R2019a) 23-Nov-2018
 * C/C++ source code generated on : Tue Apr 16 15:15:58 2019
 *
 * Target selection: ert.tlc
 * Embedded hardware selection: Specified
 * Code generation objectives: Unspecified
 * Validation result: Not run
 */

#include "rtwdemo_row_lut2d.h"

/* Exported data definition */

/* Definition for custom storage class: ExportToFile */
real_T P[3][2] = { { 1.0, 2.0 }, { 3.0, 4.0 }, { 5.0, 6.0 } } ;

/*
 * File trailer for generated code.
 *
 * [EOF]
 */

```

Code Execution Profiling with SIL and PIL

- Produce execution time metric for tasks and functions in the generated code
  - Measure execution time, self time, CPU utilization and number of calls
  - Identify tasks that require the most execution time
  - In these tasks, investigate code sections that require the most execution time



Profiling: rtwdemo_sil_topmodel

Block: CounterTypeB

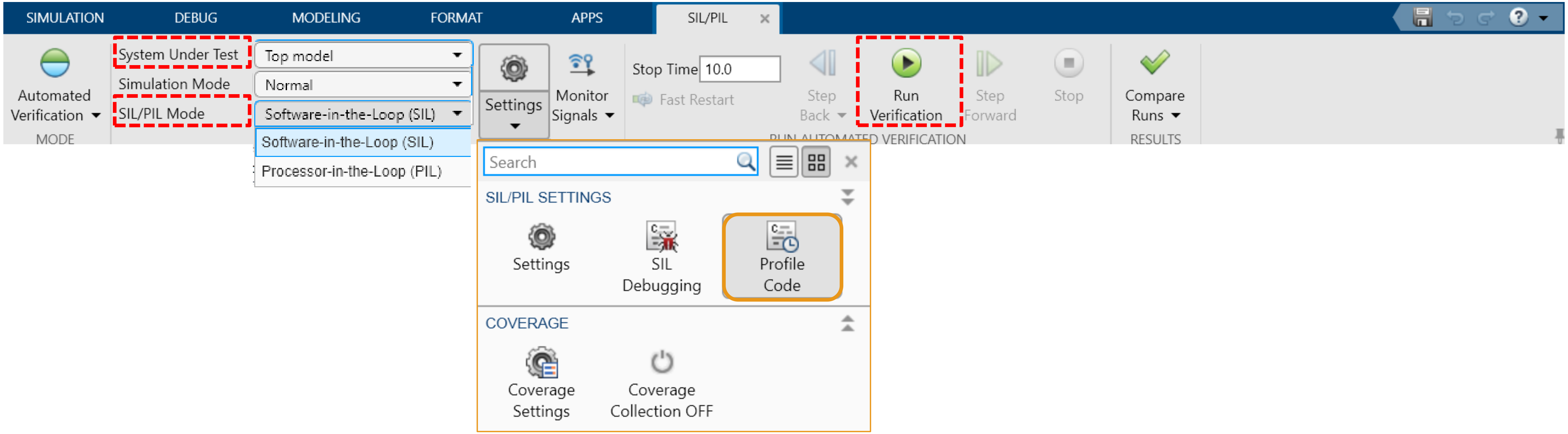
	Maximum Execution Time in ns	Average Execution Time in ns	Maximum Self Time in ns	Average Self Time in ns	Calls	
rtwdemo_sil_topmodel_initialize:CounterTypeB	13	13	13	13	1	
rtwdemo_sil_topmodel_step [0.1 0]:CounterTypeB	70	21	70	21	101	

View full code execution profiling report

Code Execution Profiling with SIL and PIL

Execution Speed

- How to Generate Execution-Time Metrics in SIL/PIL Manager



System Under Test: Top model

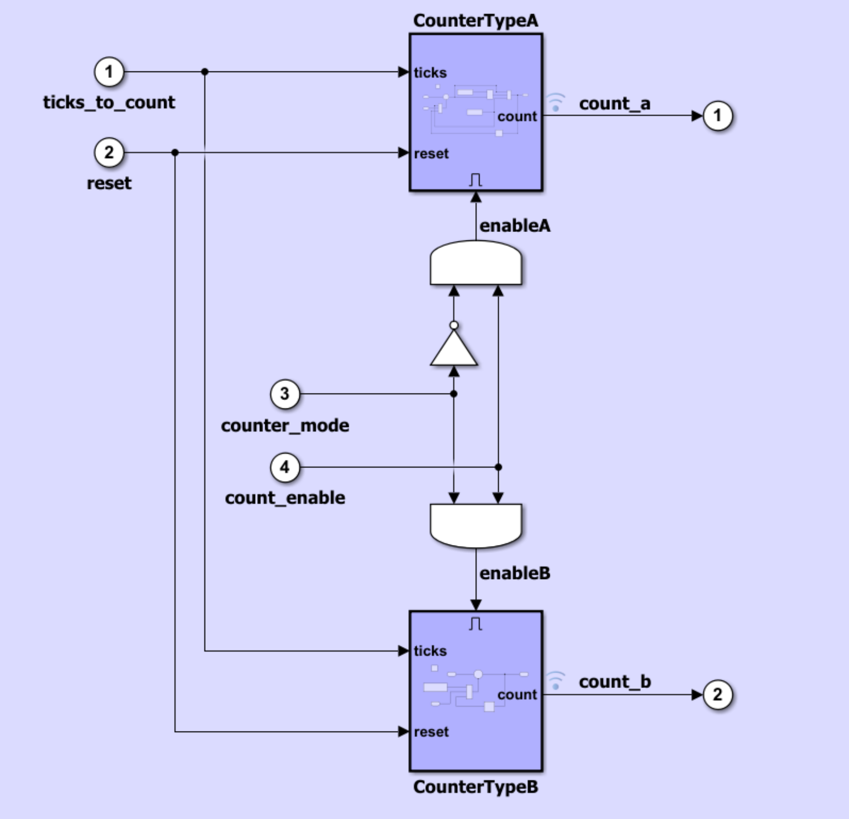
Simulation Mode: Normal

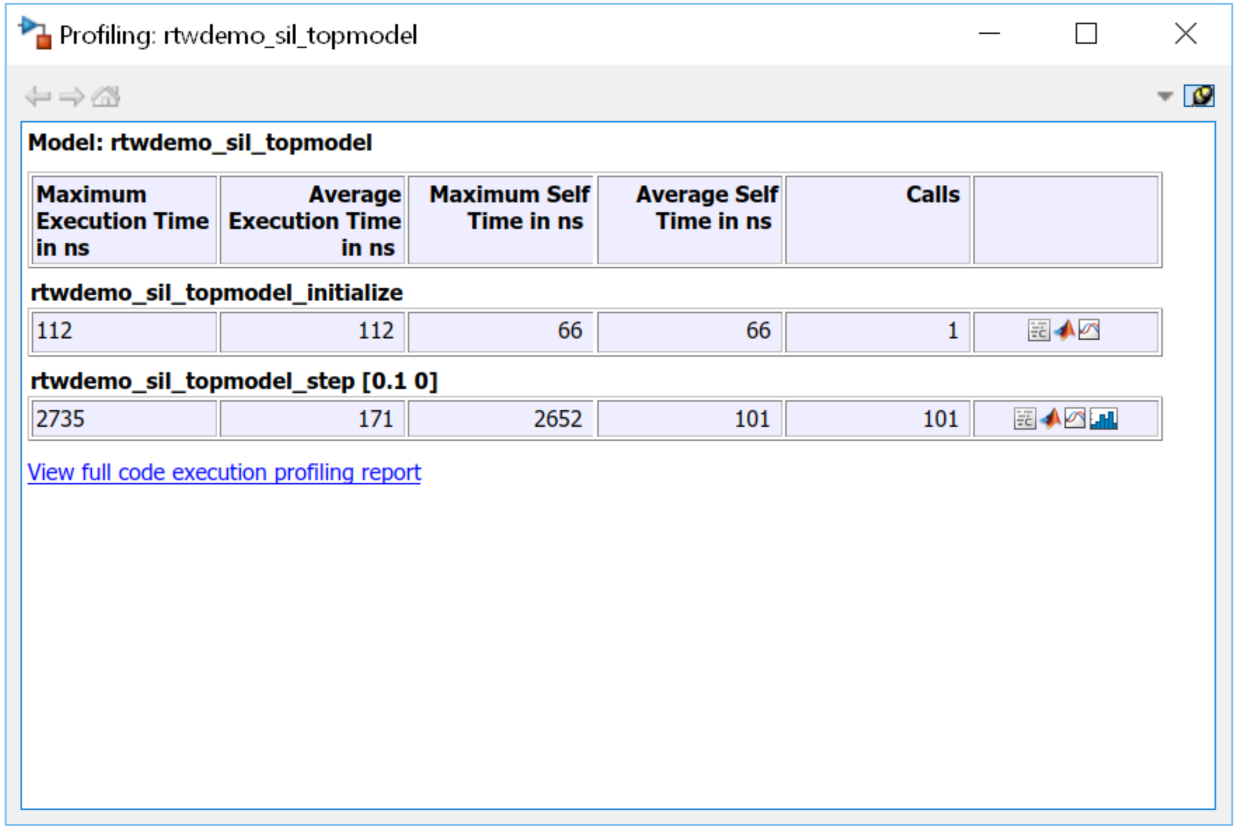
SIL/PIL Mode: Software-in-the-Loop (SIL)

Stop Time: 10.0

Improving Code and Model Performance

Execution Speed




Profiling: rtwdemo_sil_topmodel

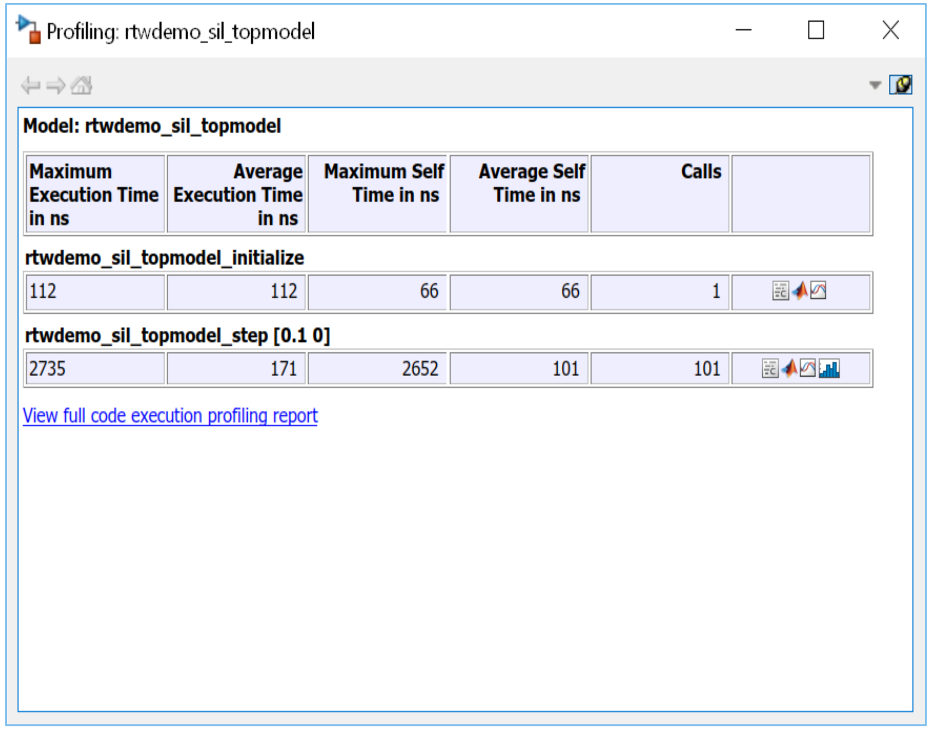
Model: rtwdemo_sil_topmodel

Maximum Execution Time in ns	Average Execution Time in ns	Maximum Self Time in ns	Average Self Time in ns	Calls	
rtwdemo_sil_topmodel_initialize					
112	112	66	66	1	
rtwdemo_sil_topmodel_step [0.1 0]					
2735	171	2652	101	101	

View full code execution profiling report

Improving Code and Model Performance

Execution Speed



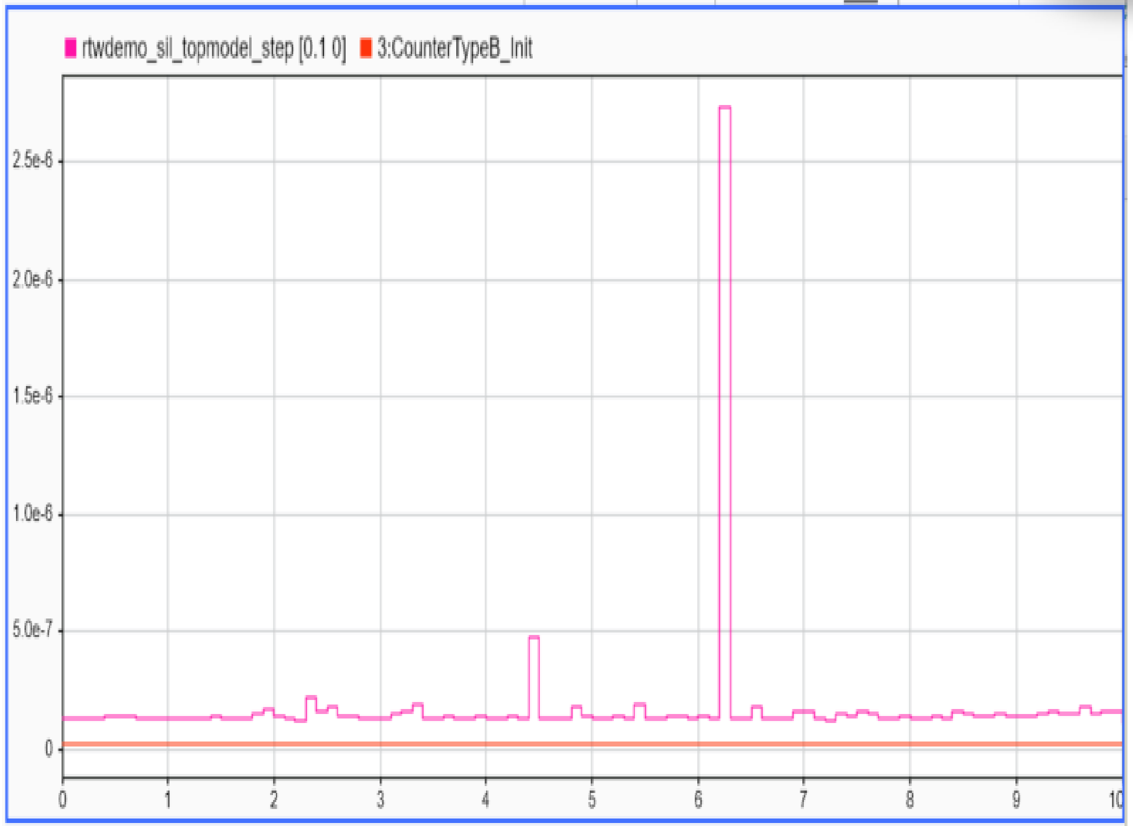
Profiling: rtwdemo_sil_topmodel

Model: rtwdemo_sil_topmodel

Maximum Execution Time in ns	Average Execution Time in ns	Maximum Self Time in ns	Average Self Time in ns	Calls	
rtwdemo_sil_topmodel_initialize					
112	112	66	66	1	
rtwdemo_sil_topmodel_step [0.1 0]					
2735	171	2652	101	101	

View full code execution profiling report

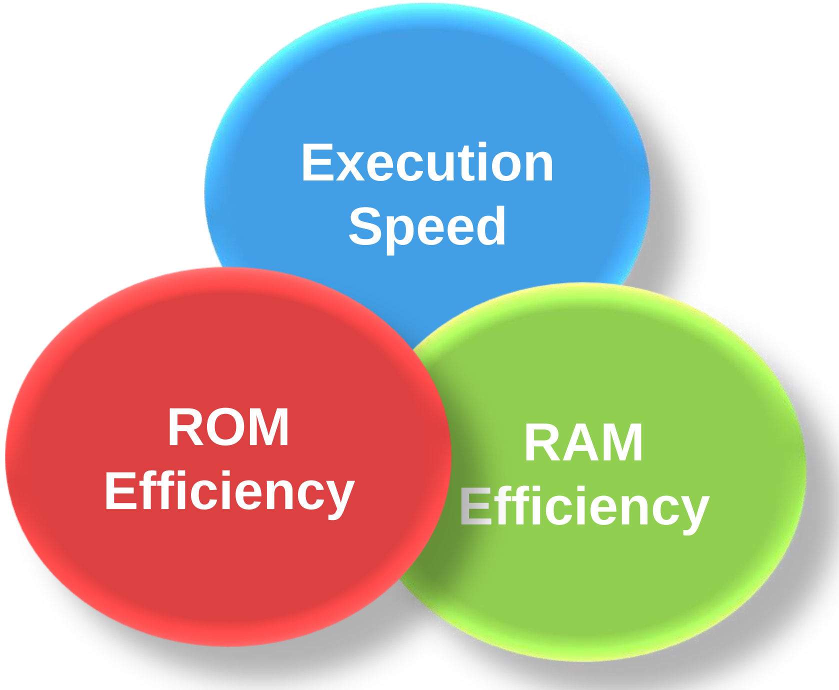
```
>> out.get('executionProfile').Sections(4)
```



Time: [101×1 double]

Key Takeaway

- Improving Modeling Patterns for Efficiency
  - Clone detection, memory efficiency
- RAM and Data Copy Reduction
  - Buffer reuse, reduction data copy
- Execution Speed
  - Row-major and column-major
  - Code execution profiling



Thank You !!