Adaptive Cruise Control System

Prepared by:
Nishantkumar V Patel

INDEX

1. General Overview

- Adaptive Cruise Control Feature for passenger cars allows the host vehicle to adapt to the speed in line with the flow of traffic. Driving in heavy traffic or keeping a safe distance to the preceding vehicle calls for a high level of concentration. The Adaptive Cruise Control feature can reduce the stress on the driver by automatically controlling the vehicle speed & maintaining a predefined minimum distance to the preceding vehicle.

- As a consequence, the driver enjoys more comfort & can concentrate on the road little better. A radar sensor is usually at the core of the Adaptive Cruise Control. Installed at the front of the vehicle, the system permanently monitors the road ahead. As long as the road ahead is clear, cruise control feature

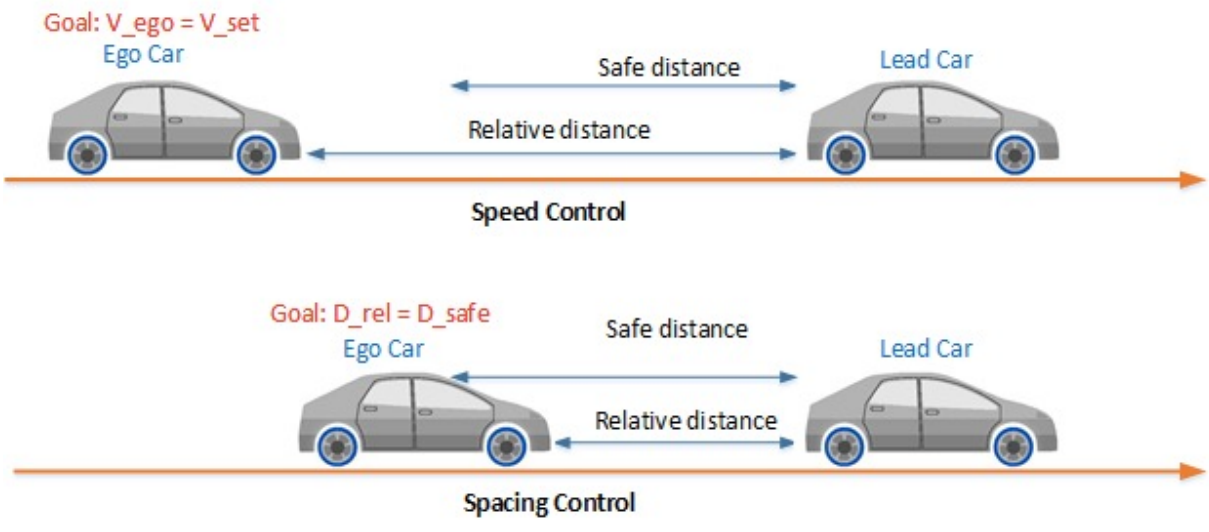


Image Courtesy: Mathworks

- maintains the speed set by the driver. If the system spots a slower vehicle within its detection range, it gently reduces speed by releasing the accelerator or actively engaging the brake control system. If the vehicle ahead speeds up or changes lanes, the cruise control automatically accelerates to the driver's desired speed.

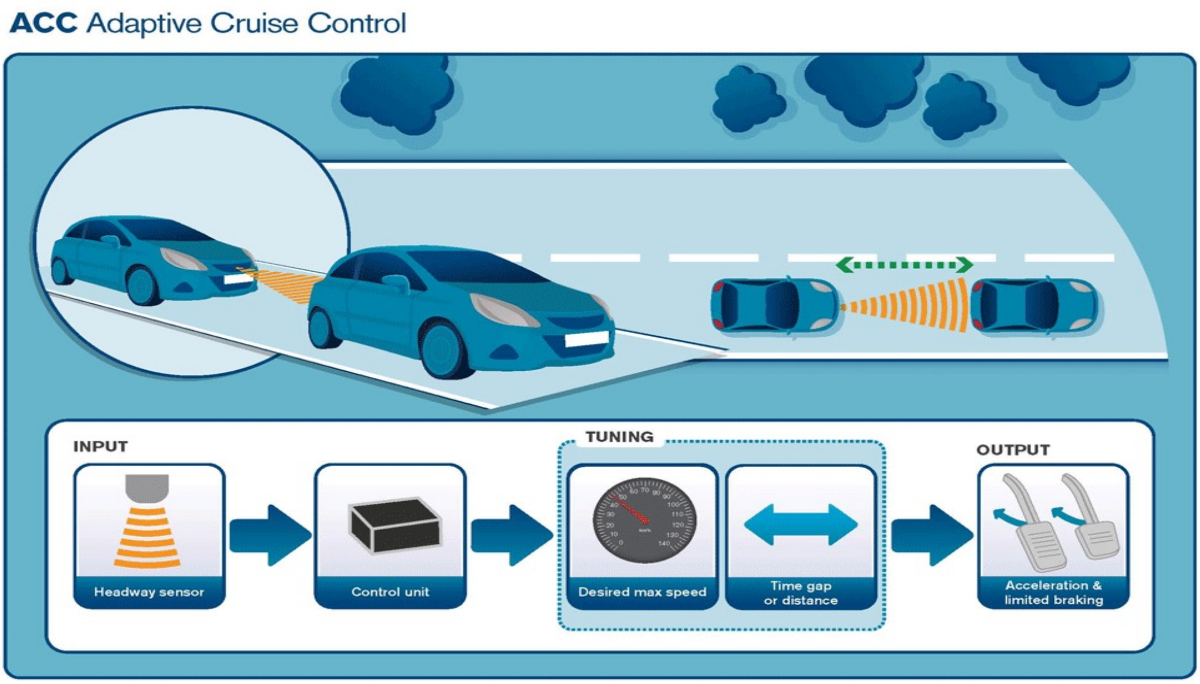


Image Courtesy: Internet

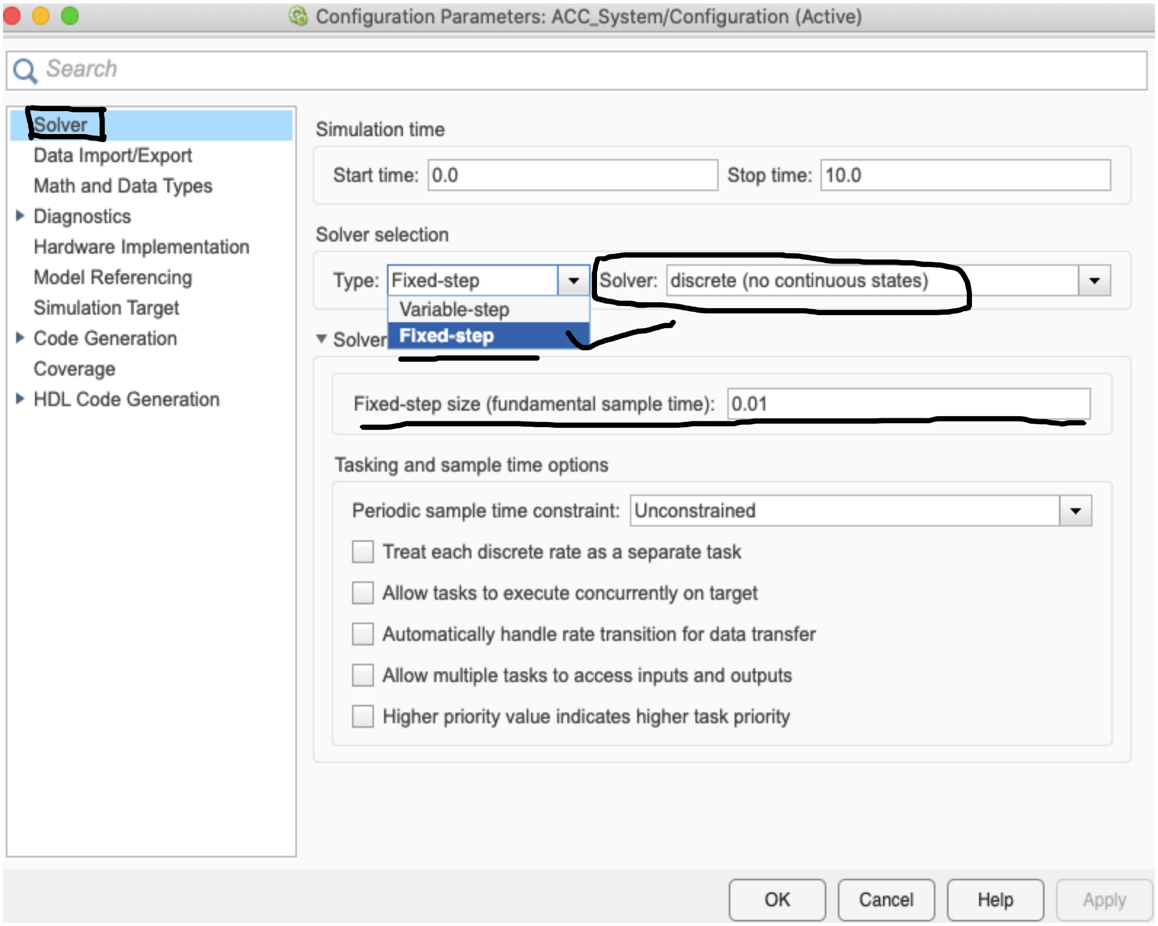
- Standard Adaptive Cruise Control can be activated from speeds of around 30 km/h (20 mph) upwards and supports the driver, primarily on cross-country journeys or on freeways. The cruise control stop & go variant is also active at speeds below 30 km/h (20 mph). It can maintain the set distance to the preceding vehicle even

- at very low speeds and can decelerate to a complete standstill. When the vehicle remains stopped longer, the driver needs only to reactivate the system, for example by briefly stepping on the gas pedal to return to cruise control mode. In this way, cruise control stop & go supports the driver even in heavy traffic and traffic jams.

2. Requirement Analysis

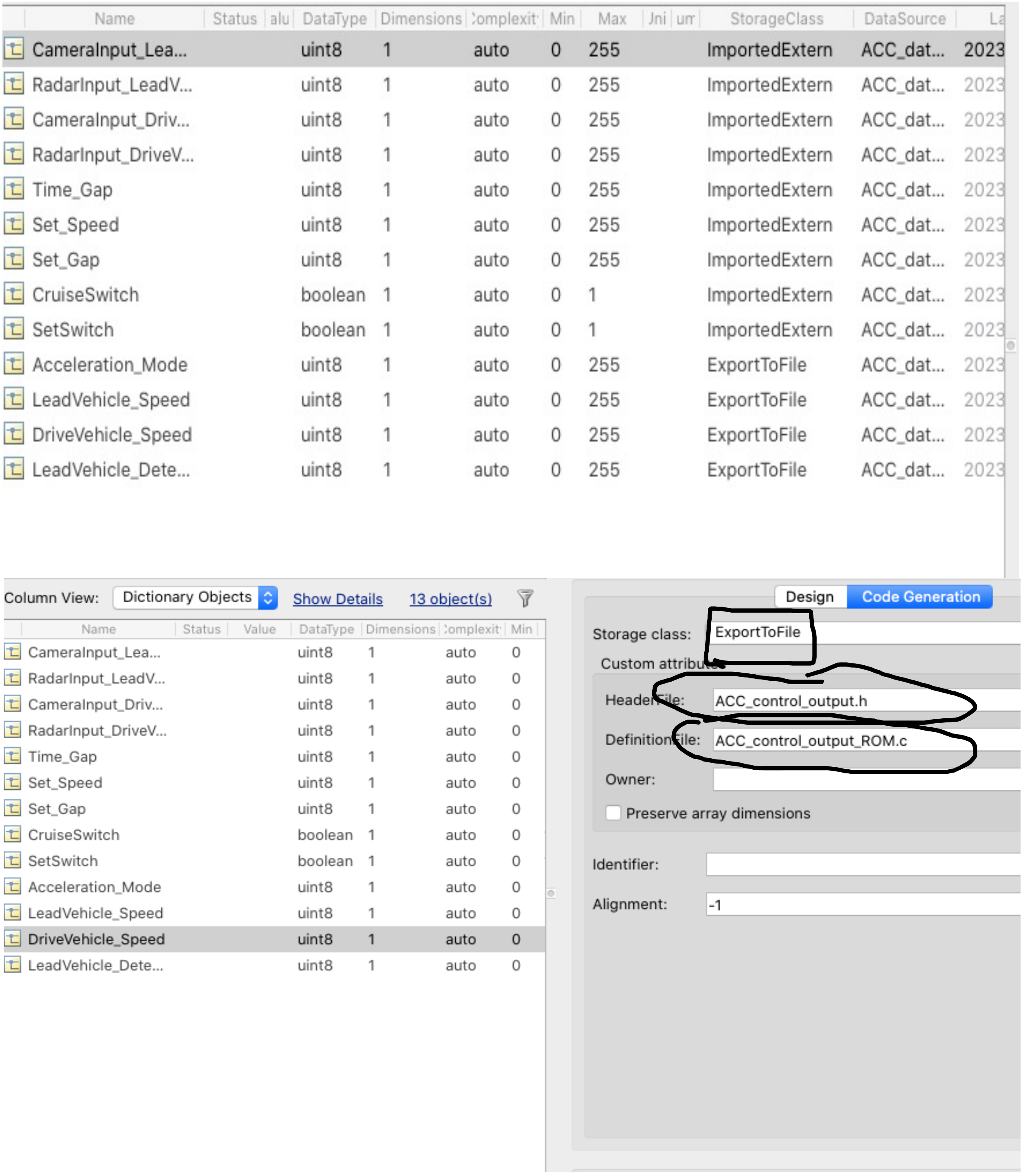
- Developing Adaptive Cruise Control feature as per the Requirement Document using MATLAB Simulink.
- Follow all the MBD related processes: Requirement Tagging & Traceability, SLDD creation, Configuration Parameter changes, Model Advisor check & Code Generation.
- In Configuration Parameters: enable “Support Floating Numbers” under Code Generation settings.
- Use Embedded Coder to generate the code.
- If choosing code generation, Storage class for Input signals: ***ImportedExtern***; Storage class for Output signal: ***Export to File***; Storage class for local signals: ***localizable***; Storage class for calibration signals: ***Const***.
- Choose sample time for all signals as 0.01s

Solver selection:



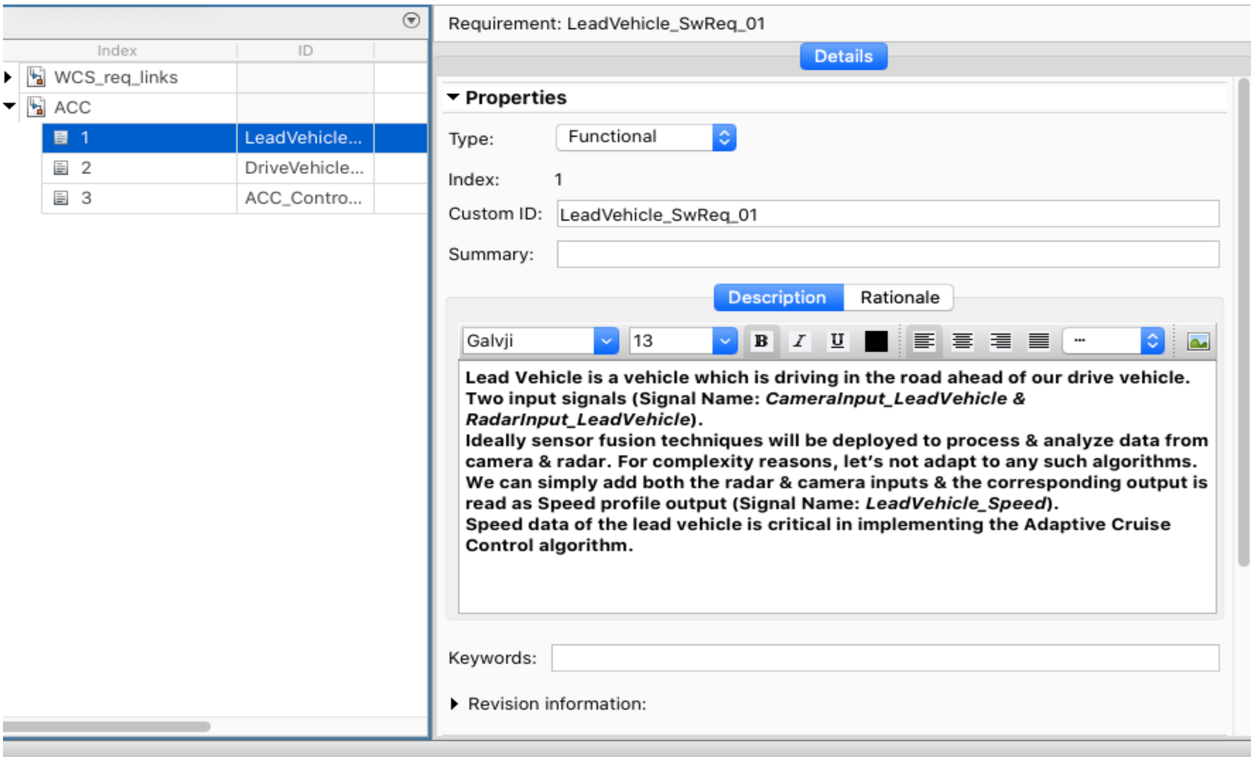
- The most common practice for model based development, code generation, and MiL-SiL verification purposes; Solver Type should be *Fixed Step* with *fundamental sample time* (as per requirement document) and Solver itself should be either *Auto selection or discrete (no continuous states)*
- Without Fixed step solver selection one is not able to generate the code nor to perform MiL/ SiL verification.

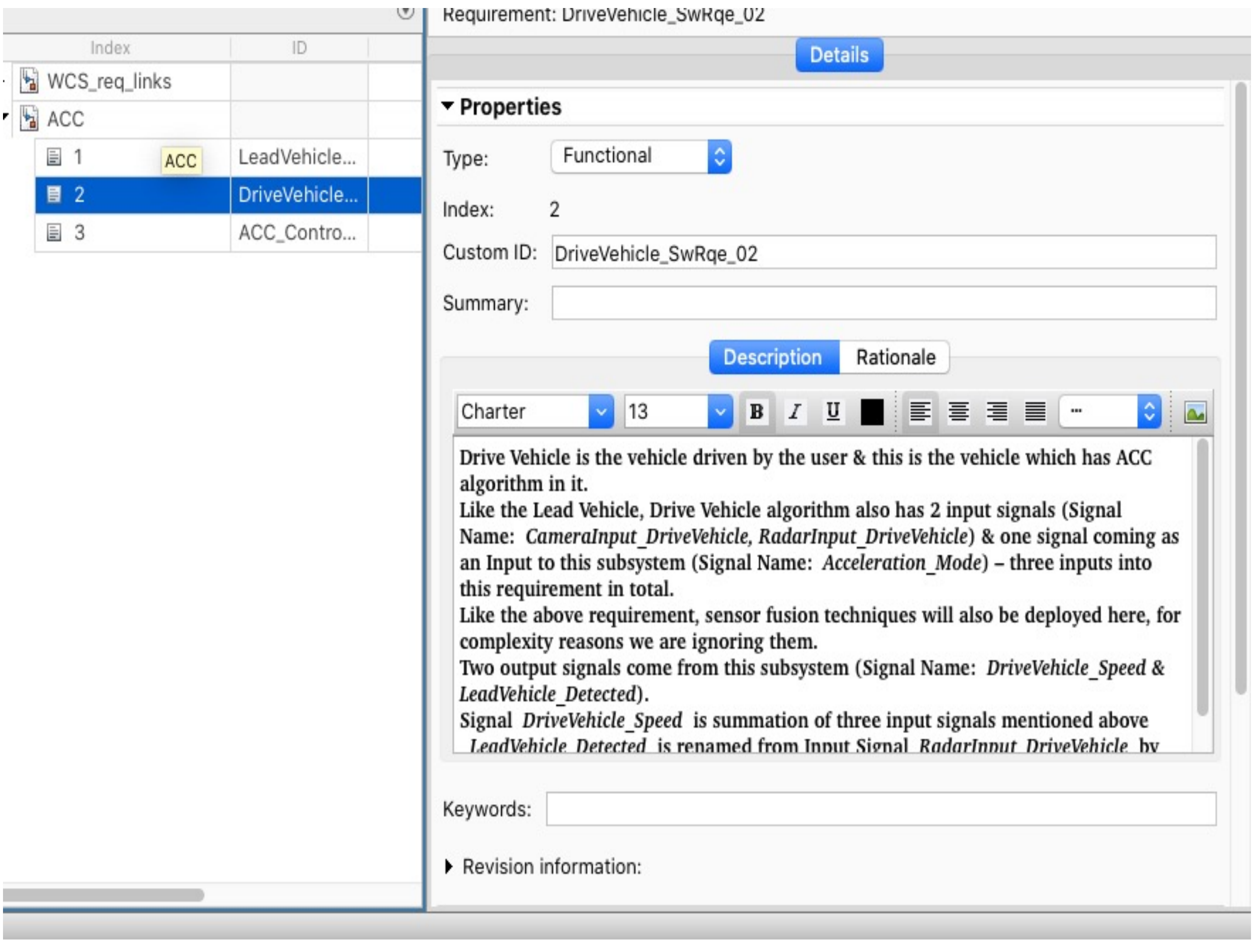
Simulink Data Dictionary:

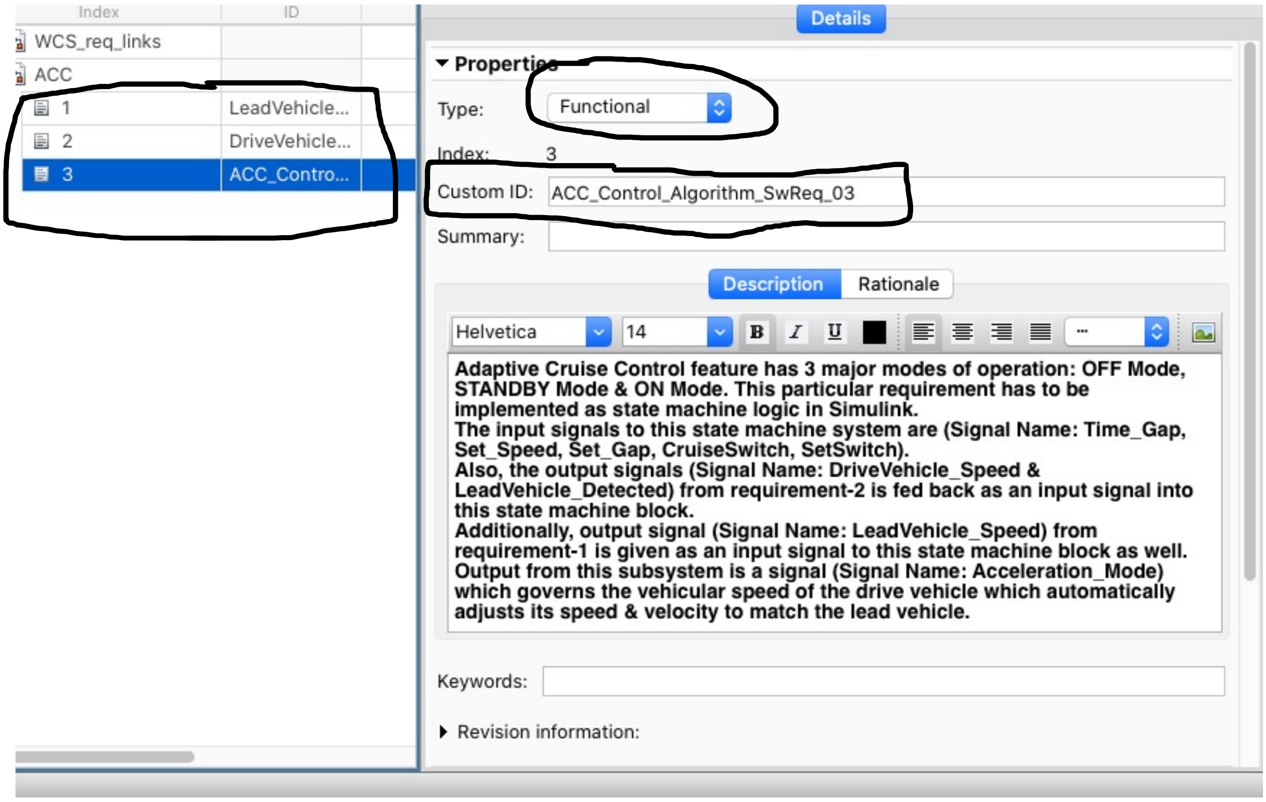


- For all output signals consisting of *ExportToFile* storage class have the same name of Header files and definition files.

Requirement Editor:







- Type of the requirement as it is default as Functional requirement. Custom ID is Software requirement ID. Each requirement has its own unique ID. Functional requirement description has been written in the section of description tab.

Requirements Report for ACC_System

Table of Contents

List of Tables

Chapter 1. Model Information for "ACC_System"

Table 1.1. ACC_System

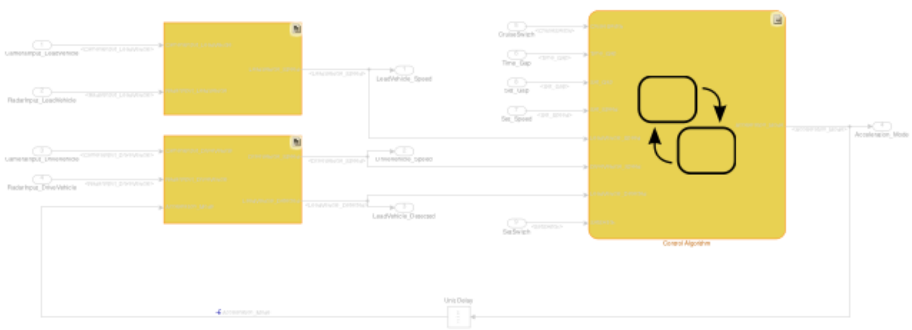
ModelVersion	1.36	*ConfigurationManager*	N/A
Created	Sat Mar 25 18:49:37 2023	*Creator*	jamesbond
LastModifiedDate	Sat Apr 01 01:23:19 2023	*LastModifiedBy*	jamesbond

Chapter 2. Traceability Summary for "ACC_System"

Table 2.1. Artifacts linked in model

ID	Artifact names stored by RMI	Last modified	# links
DOC1	ACC.slreqx	Sat Apr 01 21:04:00 2023	5

Chapter 3. System - Subsystem



Show in Simulink

Table 3.1. Objects in ACC_System/Subsystem that have Requirement Links

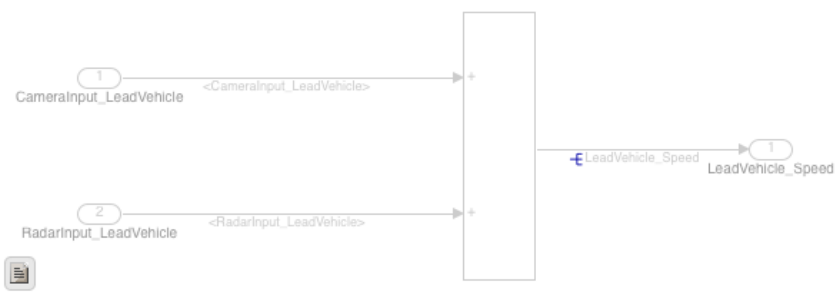
Linked Object	Requirements Data

Control Algorithm	1.	"ACC_Control_Algorithm_SwReq_03" ------- Details from ACC.slreqx: ------- Description: Adaptive Cruise Control feature has 3 major modes of operation: OFF Mode, STANDBY Mode & ON Mode. This particular requirement has to be implemented as state machine logic in Simulink. The input signals to this state machine system are (Signal Name: Time_Gap, Set_Speed, Set_Gap, CruiseSwitch, SetSwitch). Also, the output signals (Signal Name: DriveVehicle_Speed & LeadVehicle_Detected) from requirement-2 is fed back as an input signal into this state machine block. Additionally, output signal (Signal Name: LeadVehicle_Speed) from requirement-1 is given as an input signal to this state machine block as well. Output from this subsystem is a signal (Signal Name: Acceleration_Mode) which governs the vehicular speed of the drive vehicle which automatically adjusts its speed & velocity to match the lead vehicle.	DOC1, at "3"

Table 3.2. Objects in "Subsystem" that are not linked to requirements

Name	Type
Acceleration_Mode	Outport
CameraInput_DriveVehicle	Inport
CameraInput_LeadVehicle	Inport
CruiseSwitch	Inport
DriveVehicle_Speed	Outport
LeadVehicle_Detected	Outport
LeadVehicle_Speed	Outport
RadarInput_DriveVehicle	Inport
RadarInput_LeadVehicle	Inport
Set_Gap	Inport
Set_Speed	Inport
SetSwitch	Inport
Time_Gap	Inport
Unit Delay	UnitDelay

Chapter 4. System - Subsystem



Show in Simulink

Table 4.1. ACC_System/Subsystem/Subsystem Requirements Data

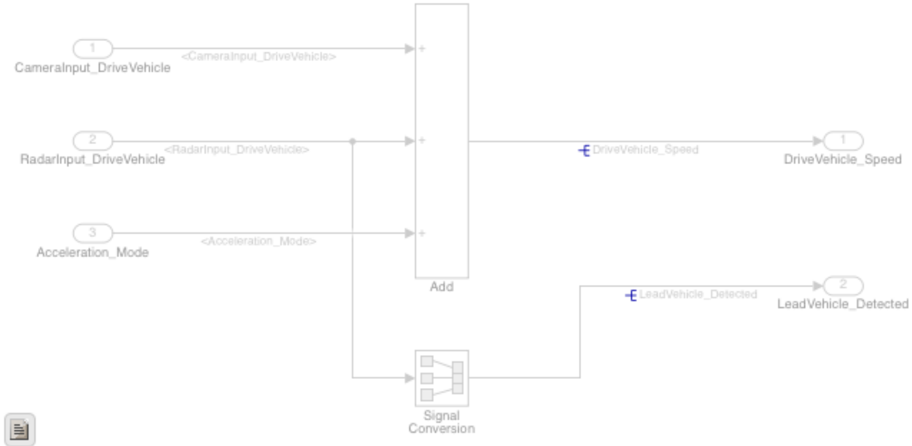
Link#	Link Description	Link Target (document name and location ID)
1.	"LeadVehicle_SwReq_01" ------- Details from ACC.slreqx: ------- Description: Lead Vehicle is a vehicle which is driving in the road ahead of our drive vehicle. Two input signals (Signal Name: CameraInput_LeadVehicle & RadarInput_LeadVehicle). Ideally sensor fusion techniques will be deployed to process & analyze data from camera & radar. For complexity reasons, let's not adapt to any such algorithms. We can simply add both the radar & camera inputs & the corresponding output is read as Speed profile output (Signal Name: LeadVehicle_Speed). Speed data of the lead vehicle is critical in implementing the Adaptive Cruise Control algorithm.	DOC1, at "1"

Table 4.2. Objects in "Subsystem" that are not linked to requirements

Name	Type
Add	Sum
CameraInput_LeadVehicle	Inport
LeadVehicle_Speed	Outport

RadarInput_LeadVehicle	Inport

Chapter 5. System - Subsystem1



[Show in Simulink]

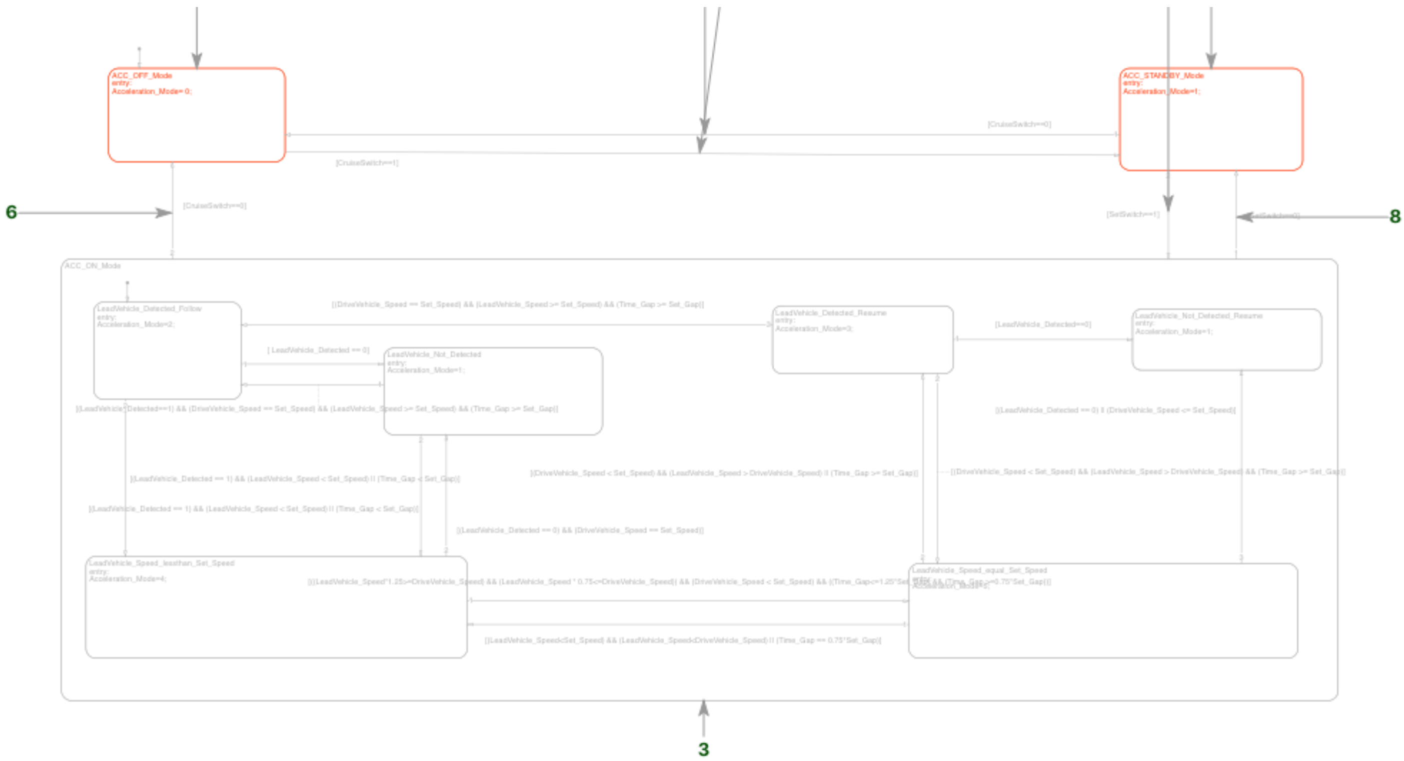
Table 5.1. ACC_System/Subsystem/Subsystem1 Requirements Data

Link#	Link Description	Link Target (document name and location ID)
1.	"DriveVehicle_SwRqe_02" ------- Details from ACC.slreqx: ------- Description: Drive Vehicle is the vehicle driven by the user & this is the vehicle which has ACC algorithm in it. Like the Lead Vehicle, Drive Vehicle algorithm also has 2 input signals (Signal Name: CameraInput_DriveVehicle, RadarInput_DriveVehicle) & one signal coming as an Input to this subsystem (Signal Name: Acceleration_Mode) – three inputs into this requirement in total. Like the above requirement, sensor fusion techniques will also be deployed here, for complexity reasons we are ignoring them. Two output signals come from this subsystem (Signal Name: DriveVehicle_Speed & LeadVehicle_Detected). Signal DriveVehicle_Speed is summation of three input signals mentioned above & LeadVehicle_Detected is renamed from Input Signal RadarInput_DriveVehicle by mere use of Signal Conversion block.	DOC1, at "2"

Table 5.2. Objects in "Subsystem1" that are not linked to requirements

Name	Type
Acceleration_Mode	Inport
Add	Sum
CameraInput_DriveVehicle	Inport
DriveVehicle_Speed	Outport
LeadVehicle_Detected	Outport
RadarInput_DriveVehicle	Inport
Signal Conversion	SignalConversion

Chapter 6. Chart - Control Algorithm



(1) [ACC_OFF_Mode]
(2) [ACC_STANDBY_Mode]
(3) ACC_ON_Mode
(4) [CruiseSwitch==0]
(5) [CruiseSwitch==1]
(6) [CruiseSwitch==0]
(7) [SetSwitch==1]
(8) [SetSwitch==0]

Table 6.1. Chart Requirements

Link#	Link Description	Link Target (document name and location ID)
1.	"ACC_Control_Algorithm_SwReq_03" ------- Details from ACC.slreqx: ------- Description: Adaptive Cruise Control feature has 3 major modes of operation: OFF Mode, STANDBY Mode & ON Mode. This particular requirement has to be implemented as state machine logic in Simulink. The input signals to this state machine system are (Signal Name: Time_Gap, Set_Speed, Set_Gap, CruiseSwitch, SetSwitch). Also, the output signals (Signal Name: DriveVehicle_Speed & LeadVehicle_Detected) from requirement-2 is fed back as an input signal into this state machine block. Additionally, output signal (Signal Name: LeadVehicle_Speed) from requirement-1 is given as an input signal to this state machine block as well. Output from this subsystem is a signal (Signal Name: Acceleration_Mode) which governs the vehicular speed of the drive vehicle which automatically adjusts its speed & velocity to match the lead vehicle.	DOC1, at "3"

Show in Simulink

Table 6.2. Objects in "Control Algorithm" that have requirements

Linked Object	Requirements

ACC_OFF_Mode	1.	"ACC_Control_Algorithm_SwReq_03" ------- Details from ACC.slreqx: ------- Description: Adaptive Cruise Control feature has 3 major modes of operation: OFF Mode, STANDBY Mode & ON Mode. This particular requirement has to be implemented as state machine logic in Simulink. The input signals to this state machine system are (Signal Name: Time_Gap, Set_Speed, Set_Gap, CruiseSwitch, SetSwitch). Also, the output signals (Signal Name: DriveVehicle_Speed & LeadVehicle_Detected) from requirement-2 is fed back as an input signal into this state machine block. Additionally, output signal (Signal Name: LeadVehicle_Speed) from requirement-1 is given as an input signal to this state machine block as well. Output from this subsystem is a signal (Signal Name: Acceleration_Mode) which governs the vehicular speed of the drive vehicle which automatically adjusts its speed & velocity to match the lead vehicle.	DOC1, at "3"
ACC_STANDBY_Mode	1.	"ACC Standby Mode state logic_03_02" ------- Details from ACC.slreqx: ------- Description: This is the second activated state inside state machine logic. Output signal Acceleration_Mode is at value 1 in this state. This state is governed by both input signals CruiseSwitch & SetSwitch. If CruiseSwitch is equal to 1, state ACC STANDBY mode will get activated. If CruiseSwitch is equal to 0, state ACC OFF mode will get activated, from either ACC ON mode or ACC STANDBY mode If SetSwitch is equal to 1, state ACC ON mode will get activated. If SetSwitch is equal to 0, state ACC STANDBY mode will get activated.	DOC1, at "7"

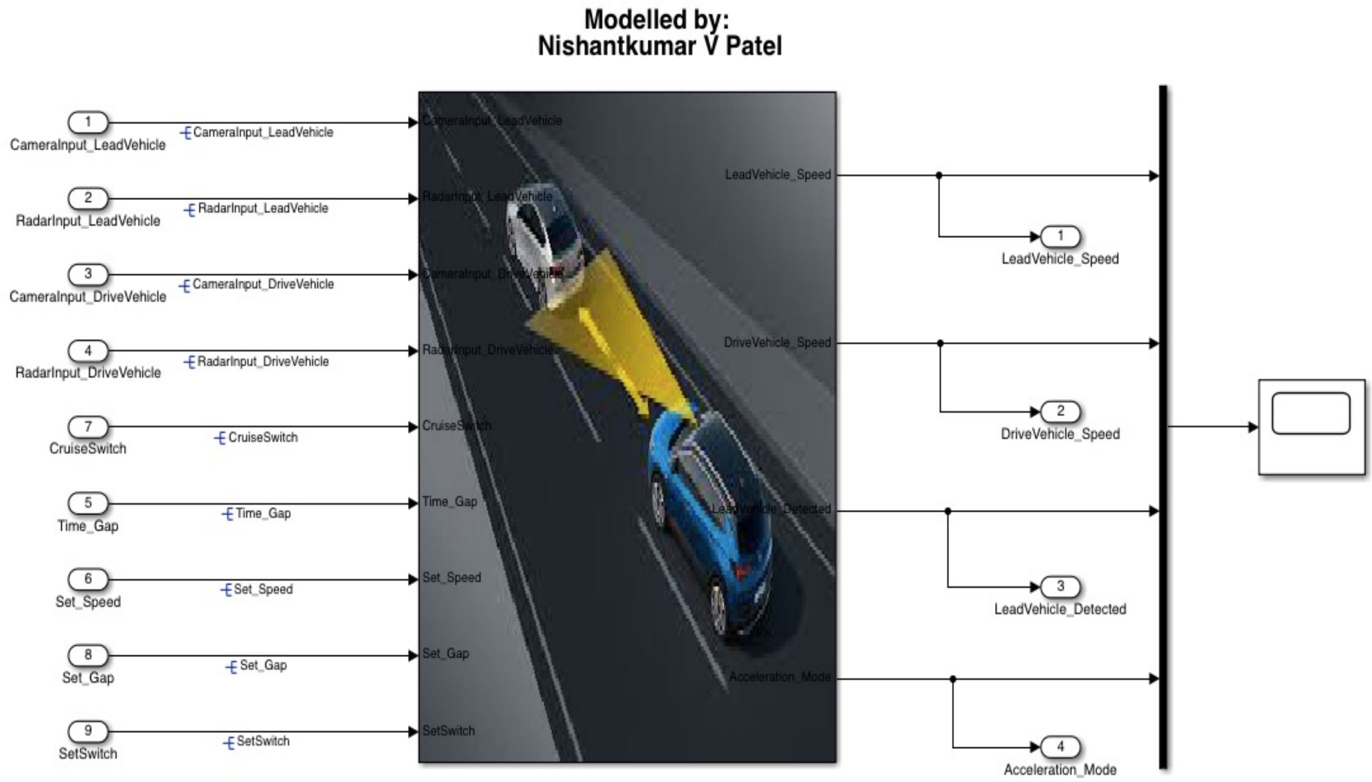
Table 6.3. Objects in "Control Algorithm" that are not linked to requirements

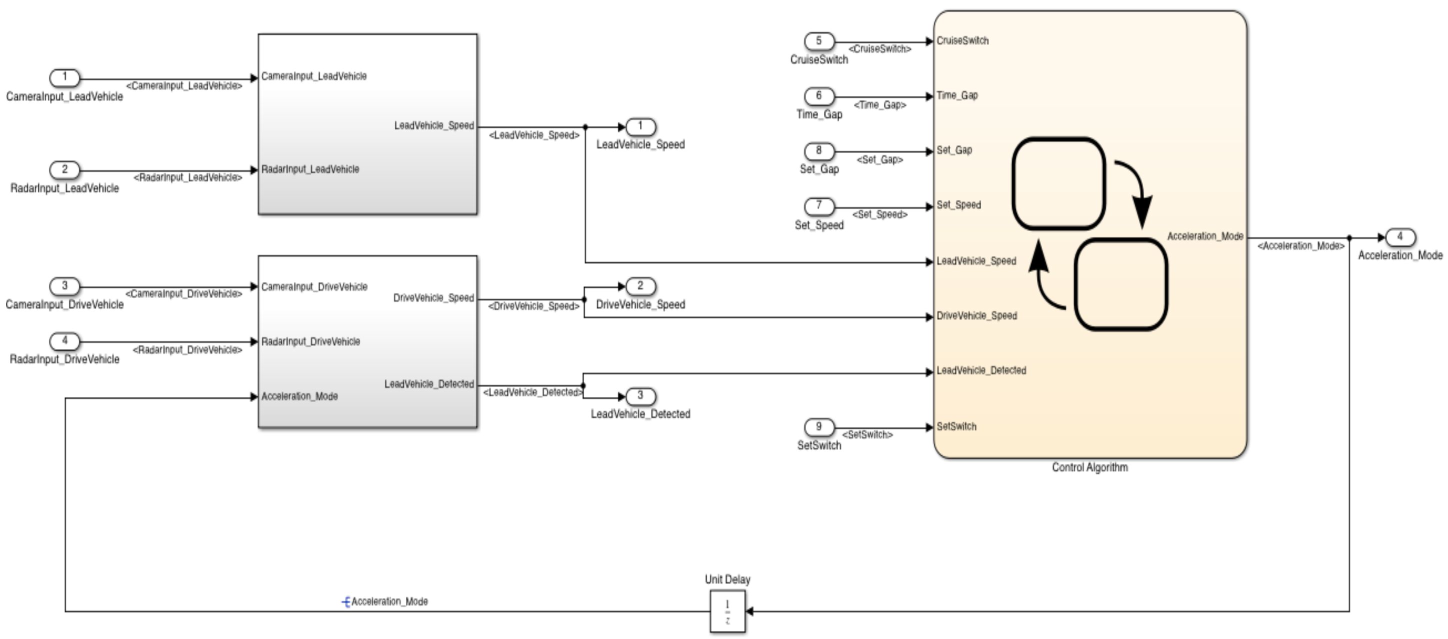
Name	Type
ACC_ON_Mode	State
Default Transition 2	Transition
[CruiseSwitch==0]	Transition
[CruiseSwitch==1]	Transition
[CruiseSwitch==0]	Transition
[SetSwitch==1]	Transition
[SetSwitch==0]	Transition
LeadVehicle_Detected_Follow	State
LeadVehicle_Detected_Resume	State
LeadVehicle_Not_Detected_Resume	State
LeadVehicle_Not_Detected	State
LeadVehicle_Speed_lessthan_Set_Speed	State
LeadVehicle_Speed_equal_Set_Speed	State
Default Transition 10	Transition
[(DriveVehicle_Speed == Set_Speed) && (LeadVehicle_Speed >= Set_Speed) && (Time_Gap >= Set_Gap)]	Transition
[LeadVehicle_Detected==0]	Transition
[LeadVehicle_Detected == 0]	Transition
[(LeadVehicle_Detected == 0) || (DriveVehicle_Speed <= Set_Speed)]	Transition
[(DriveVehicle_Speed < Set_Speed) && (LeadVehicle_Speed > DriveVehicle_Speed) || (Time_Gap >= Set_Gap)]	Transition
[(DriveVehicle_Speed < Set_Speed) && (LeadVehicle_Speed > DriveVehicle_Speed) && (Time_Gap >= Set_Gap)]	Transition
[(LeadVehicle_Detected==1) && (DriveVehicle_Speed == Set_Speed) && (LeadVehicle_Speed >= Set_Speed) && (Time_Gap >= Set_Gap)]	Transition
[(LeadVehicle_Detected == 1) && (LeadVehicle_Speed < Set_Speed) || (Time_Gap < Set_Gap)]	Transition
[(LeadVehicle_Detected == 1) && (LeadVehicle_Speed < Set_Speed) || (Time_Gap < Set_Gap)]	Transition
[(LeadVehicle_Detected == 0) && (DriveVehicle_Speed == Set_Speed)]	Transition
[((LeadVehicle_Speed*1.25>=DriveVehicle_Speed) && (LeadVehicle_Speed * 0.75<=DriveVehicle_Speed)) && (DriveVehicle_Speed < Set_Speed) && ((Time_Gap<=1.25*Set_Gap) && (Time_Gap >=0.75*Set_Gap))]	Transition
[(LeadVehicle_Speed<Set_Speed) && (LeadVehicle_Speed<DriveVehicle_Speed) || (Time_Gap == 0.75*Set_Gap)]	Transition

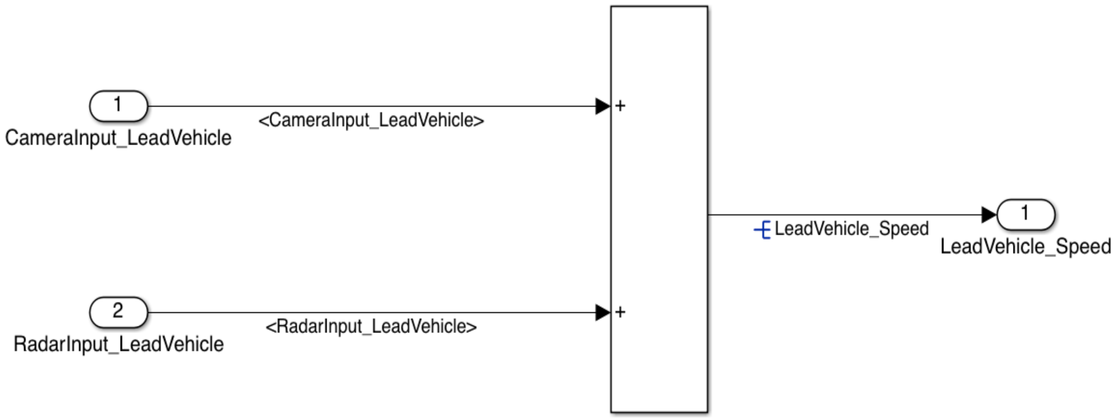
Table 7.1. Systems and subsystem blocks in "ACC_System" that have no links to requirements

Model or subsystem block	Children with links
ACC_System	None
ACC_System/Subsystem	3 out of 17

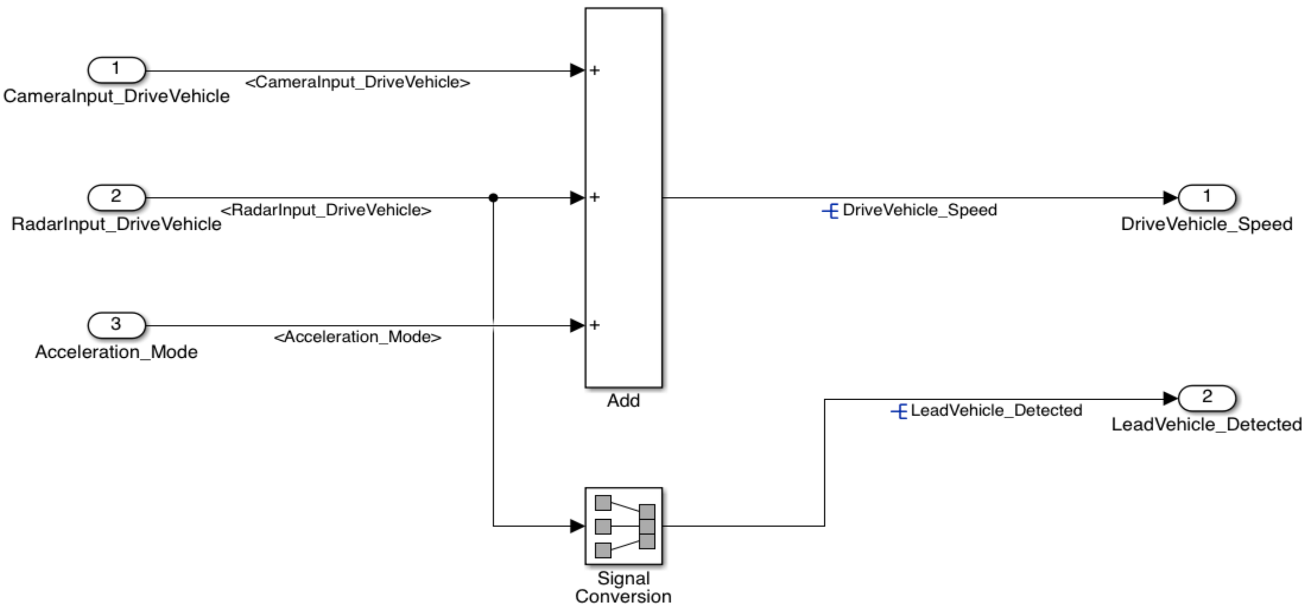
3. Model Development in Simulink



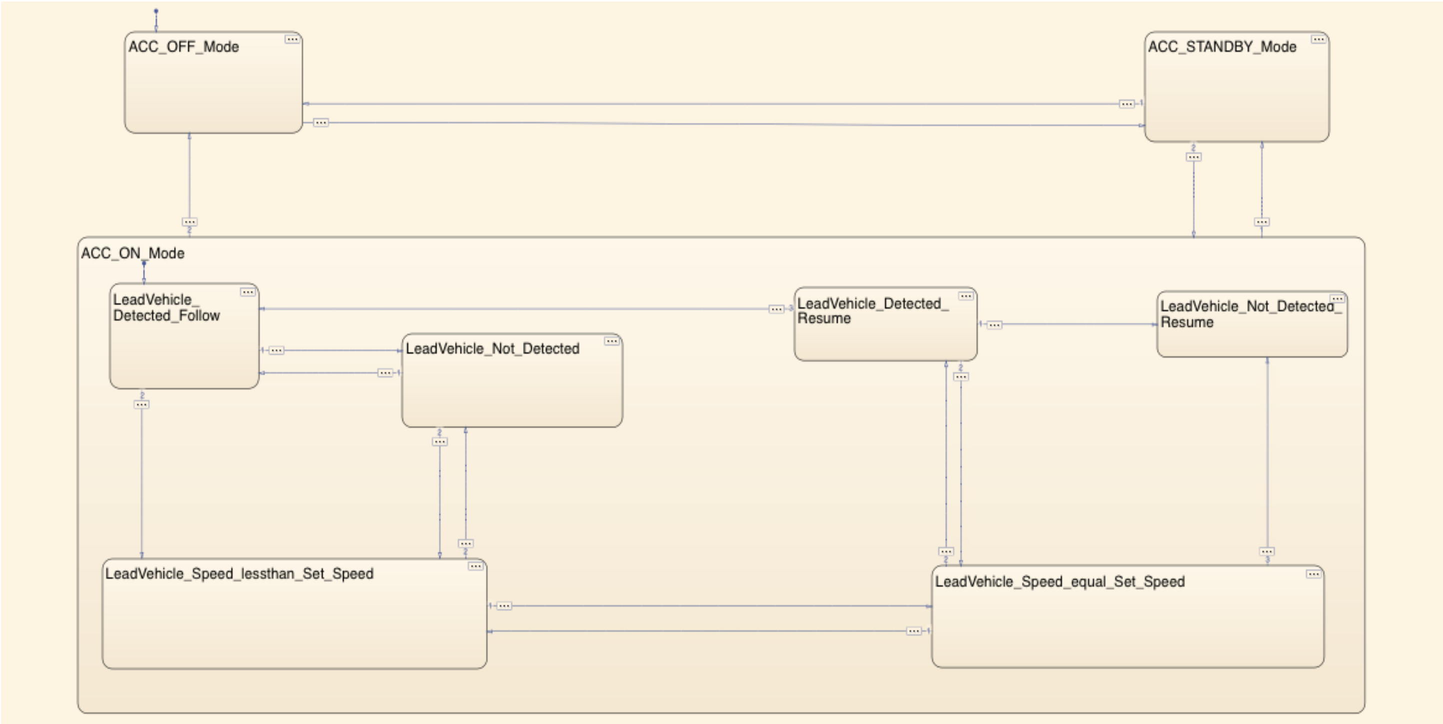




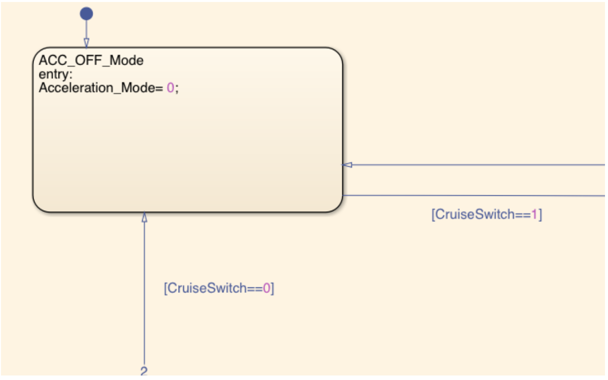
- Lead Vehicle is a vehicle which is driving in the road ahead of our drive vehicle. Two input signals (Signal Name: *CameraInput_LeadVehicle & RadarInput_LeadVehicle*).
- Ideally sensor fusion techniques will be deployed to process & analyze data from camera & radar. For complexity reasons, let's not adapt to any such algorithms.
- We can simply add both the radar & camera inputs & the corresponding output is read as Speed profile output (Signal Name: *LeadVehicle_Speed*).
- Speed data of the lead vehicle is critical in implementing the Adaptive Cruise Control algorithm.



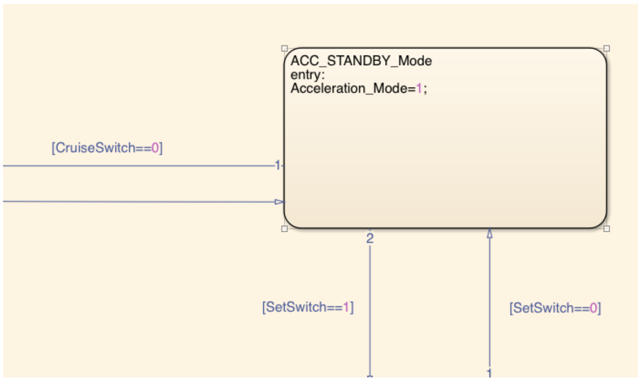
- Drive Vehicle is the vehicle driven by the user & this is the vehicle which has ACC algorithm in it.
- Like the Lead Vehicle, Drive Vehicle algorithm also has 2 input signals (Signal Name: *CameraInput_DriveVehicle, RadarInput_DriveVehicle*) & one signal coming as an Input to this subsystem (Signal Name: *Acceleration_Mode*) – three inputs into this requirement in total. Like the above requirement, sensor fusion techniques will also be deployed here, for complexity reasons we are ignoring them.
- Two output signals come from this subsystem (Signal Name: *DriveVehicle_Speed & LeadVehicle_Detected*).
- Signal *DriveVehicle_Speed* is summation of three input signals mentioned above & *LeadVehicle_Detected* is renamed from Input Signal *RadarInput_DriveVehicle* by mere use of Signal Conversion block.



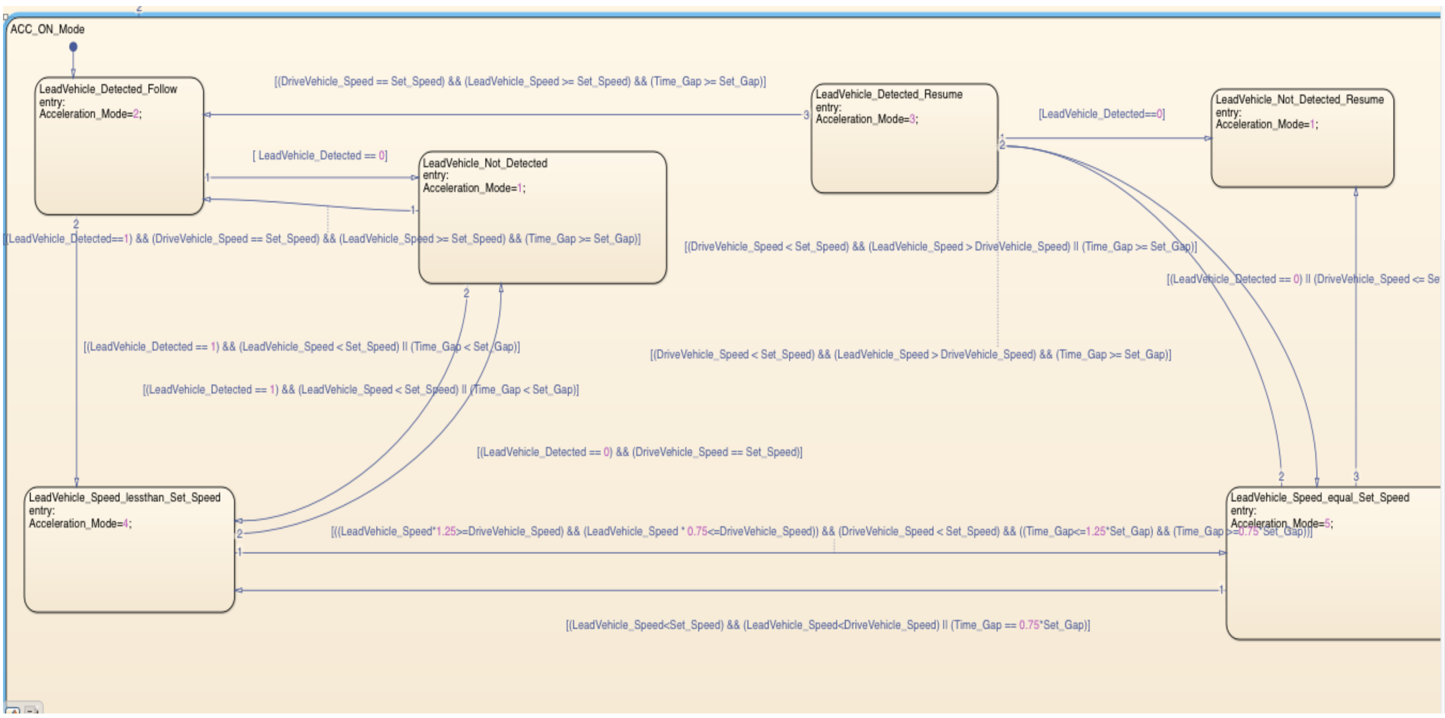
- Adaptive Cruise Control feature has 3 major modes of operation: OFF Mode, STANDBY Mode & ON Mode. This particular requirement has to be implemented as state machine logic in Simulink.
- The input signals to this state machine system are (Signal Name: *Time_Gap, Set_Speed, Set_Gap, CruiseSwitch, SetSwitch*).
- Also, the output signals (Signal Name: *DriveVehicle_Speed* & *LeadVehicle_Detected*) from requirement-2 is fed back as an input signal into this state machine block.
- Additionally, output signal (Signal Name: *LeadVehicle_Speed*) from requirement-1 is given as an input signal to this state machine block as well.
- Output from this subsystem is a signal (Signal Name: *Acceleration_Mode*) which governs the vehicular speed of the drive vehicle which automatically adjusts its speed & velocity to match the lead vehicle.



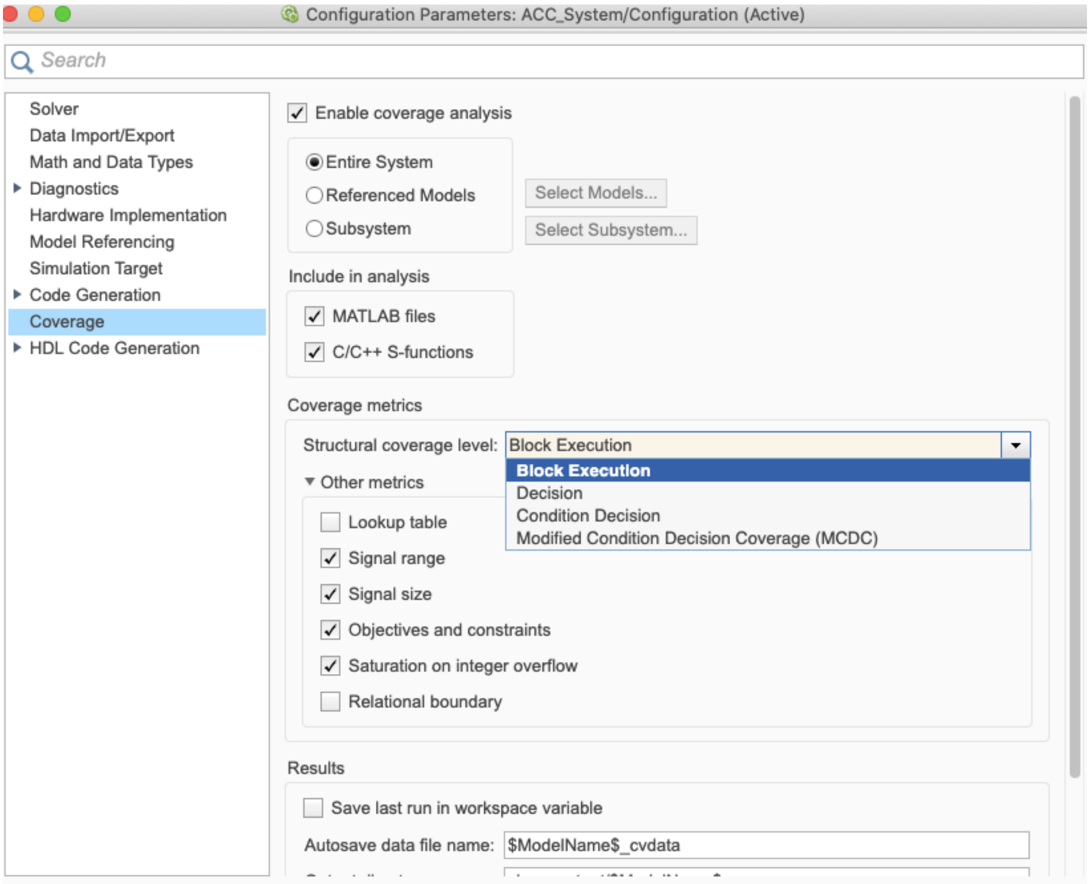
- This is the default state inside state machine logic. Output signal *Acceleration_Mode* is at value 0 in this state. This state is governed by input signal *CruiseSwitch*.
- If *CruiseSwitch* is equal to 1, state ACC STANDBY mode will get activated. If CruiseSwitch is equal to 0, state ACC OFF mode will get activated, from either *ACC ON mode* or *ACC STANDBY mode*



- This is the second activated state inside state machine logic. Output signal Acceleration_Mode is at value 1 in this state.
- This state is governed by both input signals CruiseSwitch & SetSwitch.
- If *CruiseSwitch* is equal to 1, state ACC STANDBY mode will get activated. If CruiseSwitch is equal to 0, state ACC OFF mode will get activated, from either *ACC ON mode* or *ACC STANDBY mode*
- If SetSwitch is equal to 1, state ACC ON mode will get activated. If SetSwitch is equal to 0, state ACC STANDBY mode will get activated.



4. Model Coverage Report



In model configuration settings> Coverage tab.

Check-in the Enable Coverage analysis option. Select the Structural coverage level to the model requirements. Here I have selected the Block execution. MCDC is the most common structure level is to be considered for coverage analysis

Coverage Report for ACC_System

Table of Contents

Analysis Information

Coverage Data Information

Collected in version	(R2022a)

Model Information

Model version	1.36
Author	jamesbond
Last saved	Sat Apr 01 21:07:29 2023

Simulation Optimization Options

Default parameter behavior	inlined
Block reduction	forced off
Conditional branch optimization	on

Coverage Options

Analyzed model	ACC_System
Logic block short circuiting	off

Tests

Test	Started execution	Ended execution
Run 31	02-Apr-2023 13:09:02	02-Apr-2023 13:09:12

Summary

Model Hierarchy/Complexity		Run 31	
		Saturation on integer overflow	Execution
1. ACC_System	52	0%	100%
2. . . . Subsystem	51	0%	100%
3. . . . . . . Control Algorithm	51	0%	NA
4. . . . . SF: Subsystem/Control Algorithm	50	0%	NA
5. . . . . SF: ACC_ON_Mode	43	0%	NA
6. . . . . . . Subsystem		NA	100%
7. . . . . . . Subsystem1		NA	100%

Details

1. Model "ACC_System"

Child Systems: Subsystem

Metric	Coverage (this object)	Coverage (inc. descendants)
Cyclomatic Complexity	1	52
Saturation on integer overflow	NA	0% (0/10) objective outcomes
Execution	NA	100% (4/4) objective outcomes

2. SubSystem block "Subsystem"

Justify or Exclude

Child Systems:	Control Algorithm, Subsystem, Subsystem1

Metric	Coverage (this object)	Coverage (inc. descendants)
Cyclomatic Complexity	0	51
Execution	NA	100% (4/4) objective outcomes
Saturation on integer overflow	NA	0% (0/10) objective outcomes

Full Coverage

Model Object	Metric
UnitDelay block "Unit Delay"	Execution

3. SubSystem block "Control Algorithm"

Justify or Exclude

Parent:	ACC_System/Subsystem
Child Systems:	Subsystem/Control Algorithm

Metric	Coverage (this object)	Coverage (inc. descendants)
Cyclomatic Complexity	1	51
Saturation on integer overflow	NA	0% (0/10) objective outcomes

4. Chart "Subsystem/Control Algorithm"

Justify or Exclude

Parent:	ACC_System/Subsystem/Control Algorithm
Child Systems:	ACC_ON_Mode

Metric	Coverage (this object)	Coverage (inc. descendants)
Cyclomatic Complexity	2	50
Saturation on integer overflow	NA	0% (0/10) objective outcomes

5. State "ACC_ON_Mode"

Justify or Exclude

Parent:	ACC_System/Subsystem/Control Algorithm

Metric	Coverage (this object)	Coverage (inc. descendants)
Cyclomatic Complexity	10	43
Saturation on integer overflow	NA	0% (0/10) objective outcomes

Transition "[(LeadVehicle_Speed<Set_Speed) && (LeadV..." from "LeadVehicle_Speed_equal_Set_Speed" to "LeadVehicle_Speed_lessthan_Set_Speed"

Justify or Exclude

Parent:	ACC_System/Subsystem/Control Algorithm.ACC_ON_Mode
Uncovered Links:	

Metric	Coverage
Cyclomatic Complexity	3
Saturation on integer overflow	0% (0/2) objective outcomes

```
1  [(LeadVehicle_Speed<Set_Speed) && (LeadVehicle_Speed<DriveVehicle_Speed) || (Time_Gap == 0.75*Set_Gap)]
```

#1: [(LeadVehicle_Speed<Set_Speed) && (LeadVehicle_Speed<DriveVehicle_Speed) || ...

Saturation on integer overflow analyzed

0.75*Set_Gap	0%
false	--
true	--

Justify or Exclude

Parent: ACC_System/Subsystem/Control Algorithm.ACC_ON_Mode

Uncovered Links:

Metric	Coverage
Cyclomatic Complexity	5
Saturation on integer overflow	0% (0/8) objective outcomes

```
1  [((LeadVehicle_Speed*1.25>=DriveVehicle_Speed) && (LeadVehicle_Speed * 0.75<=DriveVehicle_Speed)) && (DriveVehicle_Speed < Set_Speed)
```

#1: [((LeadVehicle_Speed*1.25>=DriveVehicle_Speed) && (LeadVehicle_Speed * 0.75<...

Saturation on integer overflow analyzed

1.25*Set_Gap	0%
false	--
true	--
0.75*Set_Gap	0%
false	--
true	--
LeadVehicle_Speed*1.25	0%
false	--
true	--
LeadVehicle_Speed * 0.75	0%
false	--
true	--

6. SubSystem block "Subsystem"

Justify or Exclude

Parent: ACC_System/Subsystem

Metric	Coverage (this object)	Coverage (inc. descendants)
Execution	NA	100% (1/1) objective outcomes

Full Coverage

Model Object	Metric
Sum block "Add"	Execution

7. SubSystem block "Subsystem1"

Justify or Exclude

Parent: ACC_System/Subsystem

Metric	Coverage (this object)	Coverage (inc. descendants)
Execution	NA	100% (2/2) objective outcomes

Full Coverage

Model Object	Metric
Sum block "Add"	Execution

Signal Ranges

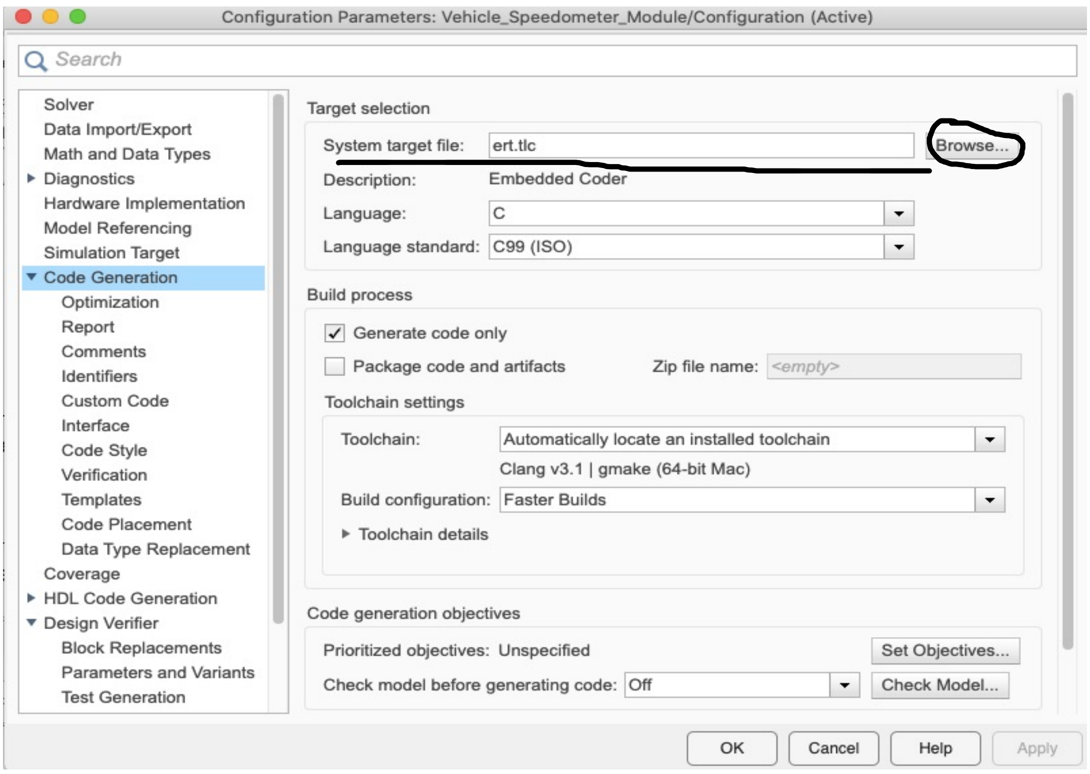
Hierarchy	Min	Max
ACC_System		
. . . Subsystem		
. . . . . . Unit Delay	0	0
. . . . . . Control Algorithm	0	0
. . . . Subsystem/Control Algorithm		
. . . . CruiseSwitch	0	0
. . . . Time_Gap	0	0
. . . . Set_Gap	0	0
. . . . Set_Speed	0	0
. . . . LeadVehicle_Speed	0	0
. . . . DriveVehicle_Speed	0	0
. . . . LeadVehicle_Detected	0	0
. . . . SetSwitch	0	0
. . . . Acceleration_Mode	0	0
. . . . . . Subsystem		
. . . . Add	0	0
. . . . . . Subsystem1		
. . . . Add	0	0
. . . . Signal Conversion	0	0

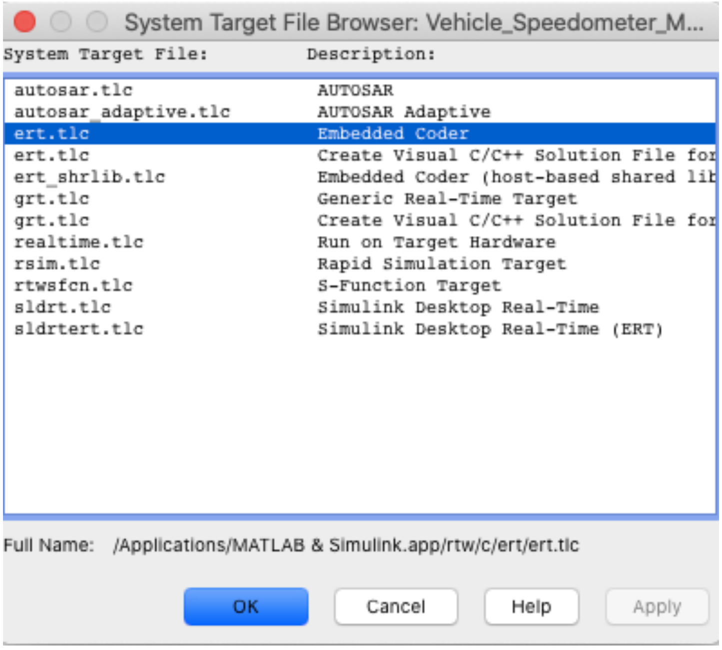
5. Code Generation

- This Code Generation Report is a C-language codings for given model with header and c-files.
- The code generation report includes number of different files.
- This code is generated based on Embedded Coder
- Here I only show main file (ert.tlc) which is simulink target file, model files (.c and .h files) so in total 3 files.
- ert.tlc (simulink target file/ main file)

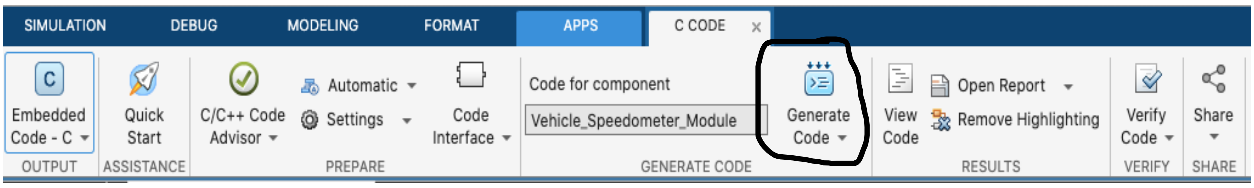
  ACC_System.c (c file/ model file)

  ACC_System.h (header file/ model file)





- open Model configuration settings>Code Generation> Target Selection tab. Click on the Browse button to access the above pop up window in order to select the System Target File.



Code Generation Report for 'ACC_System'

Model Information

Author	jamesbond
Last Modified By	jamesbond
Model Version	1.36
Tasking Mode	SingleTasking

Configuration settings at time of code generation

Code Information

System Target File	ert.tlc
Hardware Device Type	Intel->x86-64 (Windows64)
Simulink Coder Version	9.7 (R2022a) 13-Nov-2021
Timestamp of Generated Source Code	Sun Apr 2 22:49:33 2023
Location of Generated Source Code	/Users/jamesbond/MATLAB/projects/untitled1/ACC_System_ert_rtw
Type of Build	Model
Objectives Specified	**Unspecified**

Additional Information

Code Generation Advisor	Not run

ert_main.c

```c
/*
 * Academic License - for use in teaching, academic research, and meeting
 * course requirements at degree granting institutions only.  Not for
 * government, commercial, or other organizational use.
 *
 * File: ert_main.c
 *
 * Code generated for Simulink model 'ACC_System'.
 *
 * Model version                  : 1.35
 * Simulink Coder version         : 9.7 (R2022a) 13-Nov-2021
 * C/C++ source code generated on : Sat Apr  1 00:48:55 2023
 *
 * Target selection: ert.tlc
 * Embedded hardware selection: Intel->x86-64 (Windows64)
 * Code generation objectives: Unspecified
 * Validation result: Not run
 */

#include <stddef.h>
#include <stdio.h>            /* This example main program uses printf/fflush */
#include "ACC_System.h"                /* Model header file */

/*
 * Associating rt_OneStep with a real-time clock or interrupt service routine
 * is what makes the generated code "real-time".  The function rt_OneStep is
 * always associated with the base rate of the model.  Subrates are managed
 * by the base rate from inside the generated code.  Enabling/disabling
 * interrupts and floating point context switches are target specific.  This
 * example code indicates where these should take place relative to executing
 * the generated code step function.  Overrun behavior should be tailored to
 * your application needs.  This example simply sets an error status in the
 * real-time model and returns from rt_OneStep.
```

```c
 */
void rt_OneStep(void);
void rt_OneStep(void)
{
  static boolean_T OverrunFlag = false;

  /* Disable interrupts here */

  /* Check for overrun */
  if (OverrunFlag) {
    rtmSetErrorStatus(ACC_System_M, "Overrun");
    return;
  }

  OverrunFlag = true;

  /* Save FPU context here (if necessary) */
  /* Re-enable timer or interrupt here */
  /* Set model inputs here */

  /* Step the model */
  ACC_System_step();

  /* Get model outputs here */

  /* Indicate task complete */
  OverrunFlag = false;

  /* Disable interrupts here */
  /* Restore FPU context here (if necessary) */
  /* Enable interrupts here */
}

/*
 * The example main function illustrates what is required by your
```

```c
 * application code to initialize, execute, and terminate the generated code.
 * Attaching rt_OneStep to a real-time clock is target specific. This example
 * illustrates how you do this relative to initializing the model.
 */
int_T main(int_T argc, const char *argv[])
{
  /* Unused arguments */
  (void)(argc);
  (void)(argv);

  /* Initialize model */
  ACC_System_initialize();

  /* Attach rt_OneStep to a timer or interrupt service routine with
   * period 0.01 seconds (base rate of the model) here.
   * The call syntax for rt_OneStep is
   *
   *  rt_OneStep();
   */
  printf("Warning: The simulation will run forever. "
         "Generated ERT main won't simulate model step behavior. "
         "To change this behavior select the 'MAT-file logging' option.\n");
  fflush((NULL));
  while (rtmGetErrorStatus(ACC_System_M) == (NULL)) {
    /*  Perform application tasks here */
  }

  /* Terminate model */
  ACC_System_terminate();
  return 0;
}

/*
 * File trailer for generated code.
 *
```

```
 * [EOF]
 */
```

ACC_System.c

```
/*
 * Academic License - for use in teaching, academic research, and meeting
 * course requirements at degree granting institutions only.  Not for
 * government, commercial, or other organizational use.
 *
 * File: ACC_System.c
 *
 * Code generated for Simulink model 'ACC_System'.
 *
 * Model version                  : 1.35
 * Simulink Coder version         : 9.7 (R2022a) 13-Nov-2021
 * C/C++ source code generated on : Sat Apr  1 00:48:55 2023
 *
 * Target selection: ert.tlc
 * Embedded hardware selection: Intel->x86-64 (Windows64)
 * Code generation objectives: Unspecified
 * Validation result: Not run
 */

#include "ACC_System.h"
#include "rtwtypes.h"
#include "ACC_System_private.h"
#include <math.h>
#include "ACC_control_output.h"

/* Named constants for Chart: '<S1>/Control Algorithm' */
#define ACC_IN_LeadVehicle_Not_Detected ((uint8_T)3U)
#define ACC_System_IN_ACC_OFF_Mode     ((uint8_T)1U)
#define ACC_System_IN_ACC_ON_Mode      ((uint8_T)2U)
```

```c
#define ACC_System_IN_ACC_STANDBY_Mode ((uint8_T)3U)
#define ACC_System_IN_NO_ACTIVE_CHILD  ((uint8_T)0U)
#define IN_LeadVehicle_Detected_Follow ((uint8_T)1U)
#define IN_LeadVehicle_Detected_Resume ((uint8_T)2U)
#define IN_LeadVehicle_Not_Detected_Res ((uint8_T)4U)
#define IN_LeadVehicle_Speed_equal_Set_ ((uint8_T)5U)
#define IN_LeadVehicle_Speed_lessthan_S ((uint8_T)6U)

/* Block states (default storage) */
DW_ACC_System_T ACC_System_DW;

/* External outputs (root outports fed by signals with default storage) */
ExtY_ACC_System_T ACC_System_Y;

/* Real-time model */
static RT_MODEL_ACC_System_T ACC_System_M_;
RT_MODEL_ACC_System_T *const ACC_System_M = &ACC_System_M_;
real_T rt_roundd_snf(real_T u)
{
  real_T y;
  if (fabs(u) < 4.503599627370496E+15) {
    if (u >= 0.5) {
      y = floor(u + 0.5);
    } else if (u > -0.5) {
      y = u * 0.0;
    } else {
      y = ceil(u - 0.5);
    }
  } else {
    y = u;
  }

  return y;
}
```

```c
/* Model step function */
void ACC_System_step(void)
{
  /* Sum: '<S3>/Add' incorporates:
   *  Inport: '<Root>/CameraInput_LeadVehicle'
   *  Inport: '<Root>/RadarInput_LeadVehicle'
   */
  LeadVehicle_Speed = (uint8_T)(CameraInput_LeadVehicle + RadarInput_LeadVehicle);

  /* UnitDelay: '<S1>/Unit Delay' */
  Acceleration_Mode = ACC_System_Y.Acceleration_Mode_h;

  /* Sum: '<S4>/Add' incorporates:
   *  Inport: '<Root>/CameraInput_DriveVehicle'
   *  Inport: '<Root>/RadarInput_DriveVehicle'
   */
  DriveVehicle_Speed = (uint8_T)((uint8_T)(CameraInput_DriveVehicle +
    RadarInput_DriveVehicle) + Acceleration_Mode);

  /* SignalConversion: '<S4>/Signal Conversion' incorporates:
   *  Inport: '<Root>/RadarInput_DriveVehicle'
   */
  LeadVehicle_Detected = RadarInput_DriveVehicle;

  /* Chart: '<S1>/Control Algorithm' incorporates:
   *  Inport: '<Root>/CruiseSwitch'
   *  Inport: '<Root>/SetSwitch'
   *  Inport: '<Root>/Set_Gap'
   *  Inport: '<Root>/Set_Speed'
   *  Inport: '<Root>/Time_Gap'
   *  UnitDelay: '<S1>/Unit Delay'
   */
  if (ACC_System_DW.is_active_c3_ACC_System == 0U) {
    ACC_System_DW.is_active_c3_ACC_System = 1U;
    ACC_System_DW.is_c3_ACC_System = ACC_System_IN_ACC_OFF_Mode;
```

```
      ACC_System_Y.Acceleration_Mode_h = 0U;
    } else {
      switch (ACC_System_DW.is_c3_ACC_System) {
       case ACC_System_IN_ACC_OFF_Mode:
        ACC_System_Y.Acceleration_Mode_h = 0U;
        if (CruiseSwitch) {
          ACC_System_DW.is_c3_ACC_System = ACC_System_IN_ACC_STANDBY_Mode;
          ACC_System_Y.Acceleration_Mode_h = 1U;
        }
        break;

       case ACC_System_IN_ACC_ON_Mode:
        {
          if (!SetSwitch) {
            ACC_System_DW.is_ACC_ON_Mode = ACC_System_IN_NO_ACTIVE_CHILD;
            ACC_System_DW.is_c3_ACC_System = ACC_System_IN_ACC_STANDBY_Mode;
            ACC_System_Y.Acceleration_Mode_h = 1U;
          } else if (!CruiseSwitch) {
            ACC_System_DW.is_ACC_ON_Mode = ACC_System_IN_NO_ACTIVE_CHILD;
            ACC_System_DW.is_c3_ACC_System = ACC_System_IN_ACC_OFF_Mode;
            ACC_System_Y.Acceleration_Mode_h = 0U;
          } else {
            switch (ACC_System_DW.is_ACC_ON_Mode) {
             case IN_LeadVehicle_Detected_Follow:
              ACC_System_Y.Acceleration_Mode_h = 2U;
              if (LeadVehicle_Detected == 0) {
                ACC_System_DW.is_ACC_ON_Mode = ACC_IN_LeadVehicle_Not_Detected;
                ACC_System_Y.Acceleration_Mode_h = 1U;
              } else if (((LeadVehicle_Detected == 1) && (LeadVehicle_Speed <
                           Set_Speed)) || (Time_Gap < Set_Gap)) {
                ACC_System_DW.is_ACC_ON_Mode = IN_LeadVehicle_Speed_lessthan_S;
                ACC_System_Y.Acceleration_Mode_h = 4U;
              }
              break;
```

```c
case IN_LeadVehicle_Detected_Resume:
 ACC_System_Y.Acceleration_Mode_h = 3U;
 if (LeadVehicle_Detected == 0) {
   ACC_System_DW.is_ACC_ON_Mode = IN_LeadVehicle_Not_Detected_Res;
   ACC_System_Y.Acceleration_Mode_h = 1U;
 } else if ((DriveVehicle_Speed < Set_Speed) && (LeadVehicle_Speed >
            DriveVehicle_Speed) && (Time_Gap >= Set_Gap)) {
   ACC_System_DW.is_ACC_ON_Mode = IN_LeadVehicle_Speed_equal_Set_;
   ACC_System_Y.Acceleration_Mode_h = 5U;
 } else if ((DriveVehicle_Speed == Set_Speed) && (LeadVehicle_Speed >=
            Set_Speed) && (Time_Gap >= Set_Gap)) {
   ACC_System_DW.is_ACC_ON_Mode = IN_LeadVehicle_Detected_Follow;
   ACC_System_Y.Acceleration_Mode_h = 2U;
 }
 break;

case ACC_IN_LeadVehicle_Not_Detected:
 ACC_System_Y.Acceleration_Mode_h = 1U;
 if ((LeadVehicle_Detected == 1) && (DriveVehicle_Speed == Set_Speed)
     && (LeadVehicle_Speed >= Set_Speed) && (Time_Gap >= Set_Gap)) {
   ACC_System_DW.is_ACC_ON_Mode = IN_LeadVehicle_Detected_Follow;
   ACC_System_Y.Acceleration_Mode_h = 2U;
 } else if (((LeadVehicle_Detected == 1) && (LeadVehicle_Speed <
             Set_Speed)) || (Time_Gap < Set_Gap)) {
   ACC_System_DW.is_ACC_ON_Mode = IN_LeadVehicle_Speed_lessthan_S;
   ACC_System_Y.Acceleration_Mode_h = 4U;
 }
 break;

case IN_LeadVehicle_Not_Detected_Res:
 ACC_System_Y.Acceleration_Mode_h = 1U;
 break;

case IN_LeadVehicle_Speed_equal_Set_:
 ACC_System_Y.Acceleration_Mode_h = 5U;
```

```
if (((LeadVehicle_Speed < Set_Speed) && (LeadVehicle_Speed <
      DriveVehicle_Speed)) || ((int32_T)rt_roundd_snf(0.75 * (real_T)
      Set_Gap) == Time_Gap)) {
  ACC_System_DW.is_ACC_ON_Mode = IN_LeadVehicle_Speed_lessthan_S;
  ACC_System_Y.Acceleration_Mode_h = 4U;
} else if (((DriveVehicle_Speed < Set_Speed) && (LeadVehicle_Speed >
  DriveVehicle_Speed)) || (Time_Gap >= Set_Gap)) {
  ACC_System_DW.is_ACC_ON_Mode = IN_LeadVehicle_Detected_Resume;
  ACC_System_Y.Acceleration_Mode_h = 3U;
} else if ((LeadVehicle_Detected == 0) || (DriveVehicle_Speed <=
            Set_Speed)) {
  ACC_System_DW.is_ACC_ON_Mode = IN_LeadVehicle_Not_Detected_Res;
  ACC_System_Y.Acceleration_Mode_h = 1U;
}
break;

default:
{
  int32_T tmp;
  int32_T tmp_0;
  uint8_T tmp_1;
  uint8_T tmp_2;

  /* case IN_LeadVehicle_Speed_lessthan_Set_Speed: */
  ACC_System_Y.Acceleration_Mode_h = 4U;
  tmp = (int32_T)rt_roundd_snf((real_T)LeadVehicle_Speed * 1.25);
  tmp_0 = (int32_T)rt_roundd_snf(1.25 * (real_T)Set_Gap);
  if (tmp < 256) {
    tmp_1 = (uint8_T)tmp;
  } else {
    tmp_1 = MAX_uint8_T;
  }

  if (tmp_0 < 256) {
    tmp_2 = (uint8_T)tmp_0;
```

```
        } else {
          tmp_2 = MAX_uint8_T;
        }

        if ((tmp_1 >= DriveVehicle_Speed) && ((int32_T)rt_roundd_snf
             ((real_T)LeadVehicle_Speed * 0.75) <= DriveVehicle_Speed) &&
            (DriveVehicle_Speed < Set_Speed) && (Time_Gap <= tmp_2) &&
            (Time_Gap >= (int32_T)rt_roundd_snf(0.75 * (real_T)Set_Gap)))
        {
          ACC_System_DW.is_ACC_ON_Mode = IN_LeadVehicle_Speed_equal_Set_;
          ACC_System_Y.Acceleration_Mode_h = 5U;
        } else if ((LeadVehicle_Detected == 0) && (DriveVehicle_Speed ==
                    Set_Speed)) {
          ACC_System_DW.is_ACC_ON_Mode = ACC_IN_LeadVehicle_Not_Detected;
          ACC_System_Y.Acceleration_Mode_h = 1U;
        }
      }
      break;
    }
  }
}
break;

default:
 /* case IN_ACC_STANDBY_Mode: */
 ACC_System_Y.Acceleration_Mode_h = 1U;
 if (!CruiseSwitch) {
   ACC_System_DW.is_c3_ACC_System = ACC_System_IN_ACC_OFF_Mode;
   ACC_System_Y.Acceleration_Mode_h = 0U;
 } else if (SetSwitch) {
   ACC_System_DW.is_c3_ACC_System = ACC_System_IN_ACC_ON_Mode;
   ACC_System_DW.is_ACC_ON_Mode = IN_LeadVehicle_Detected_Follow;
   ACC_System_Y.Acceleration_Mode_h = 2U;
 }
 break;
```

```
    }
  }

  /* End of Chart: '<S1>/Control Algorithm' */
}

/* Model initialize function */
void ACC_System_initialize(void)
{
  /* (no initialization code required) */
}

/* Model terminate function */
void ACC_System_terminate(void)
{
  /* (no terminate code required) */
}

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

ACC_System.h

```c
/*
 * Academic License - for use in teaching, academic research, and meeting
 * course requirements at degree granting institutions only.  Not for
 * government, commercial, or other organizational use.
 *
 * File: ACC_System.h
 *
 * Code generated for Simulink model 'ACC_System'.
 *
 * Model version                  : 1.35
 * Simulink Coder version         : 9.7 (R2022a) 13-Nov-2021
 * C/C++ source code generated on : Sat Apr  1 00:48:55 2023
 *
 * Target selection: ert.tlc
 * Embedded hardware selection: Intel->x86-64 (Windows64)
 * Code generation objectives: Unspecified
 * Validation result: Not run
 */

#ifndef RTW_HEADER_ACC_System_h_
#define RTW_HEADER_ACC_System_h_
#ifndef ACC_System_COMMON_INCLUDES_
#define ACC_System_COMMON_INCLUDES_
#include "rtwtypes.h"
#endif                                 /* ACC_System_COMMON_INCLUDES_ */
```

```c
#include "ACC_System_types.h"

/* Includes for objects with custom storage classes */
#include "ACC_control_output.h"

/* Macros for accessing real-time model data structure */
#ifndef rtmGetErrorStatus
#define rtmGetErrorStatus(rtm)         ((rtm)->errorStatus)
#endif

#ifndef rtmSetErrorStatus
#define rtmSetErrorStatus(rtm, val)    ((rtm)->errorStatus = (val))
#endif

/* Block states (default storage) for system '<Root>' */
typedef struct {
  uint8_T is_active_c3_ACC_System;     /* '<S1>/Control Algorithm' */
  uint8_T is_c3_ACC_System;            /* '<S1>/Control Algorithm' */
  uint8_T is_ACC_ON_Mode;              /* '<S1>/Control Algorithm' */
} DW_ACC_System_T;

/* External outputs (root outports fed by signals with default storage) */
typedef struct {
  uint8_T Acceleration_Mode_h;         /* '<Root>/Acceleration_Mode' */
} ExtY_ACC_System_T;

/* Real-time Model Data Structure */
struct tag_RTM_ACC_System_T {
  const char_T * volatile errorStatus;
};

/* Block states (default storage) */
extern DW_ACC_System_T ACC_System_DW;

/* External outputs (root outports fed by signals with default storage) */
```

```c
extern ExtY_ACC_System_T ACC_System_Y;

/* Model entry point functions */
extern void ACC_System_initialize(void);
extern void ACC_System_step(void);
extern void ACC_System_terminate(void);

/* Real-time Model object */
extern RT_MODEL_ACC_System_T *const ACC_System_M;

/*-
 * These blocks were eliminated from the model due to optimizations:
 *
 * Block '<Root>/Scope' : Unused code path elimination
 */

/*-
 * The generated code includes comments that allow you to trace directly
 * back to the appropriate location in the model.  The basic format
 * is <system>/block_name, where system is the system number (uniquely
 * assigned by Simulink) and block_name is the name of the block.
 *
 * Use the MATLAB hilite_system command to trace the generated code back
 * to the model.  For example,
 *
 * hilite_system('<S3>')    - opens system 3
 * hilite_system('<S3>/Kp') - opens and selects block Kp which resides in S3
 *
 * Here is the system hierarchy for this model
 *
 * '<Root>' : 'ACC_System'
 * '<S1>'   : 'ACC_System/Subsystem'
 * '<S2>'   : 'ACC_System/Subsystem/Control Algorithm'
 * '<S3>'   : 'ACC_System/Subsystem/Subsystem'
 * '<S4>'   : 'ACC_System/Subsystem/Subsystem1'
```

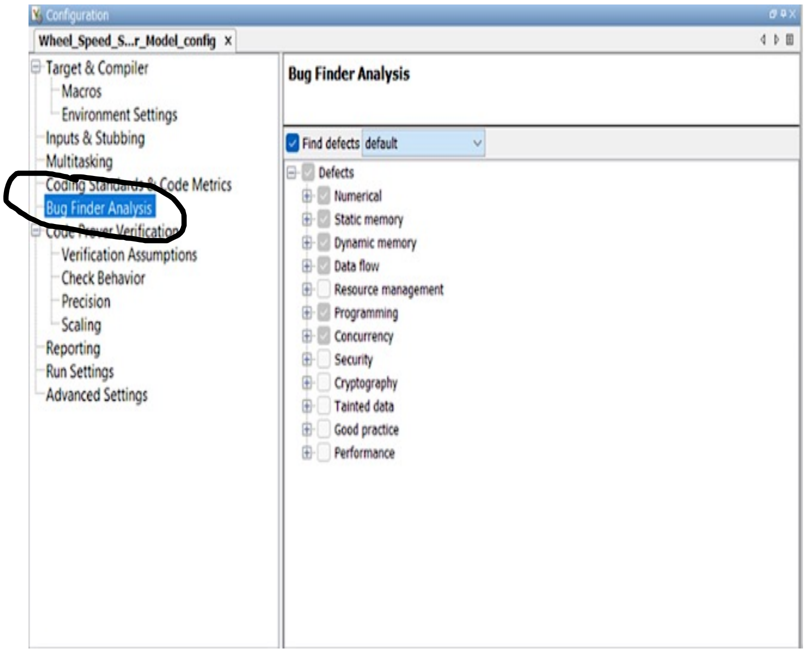
```
 */
#endif                                 /* RTW_HEADER_ACC_System_h_ */

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

Users/jamesbond/Desktop/ACC_System_ert_rtw/ACC_System.h/
Ln25Col39

has popup

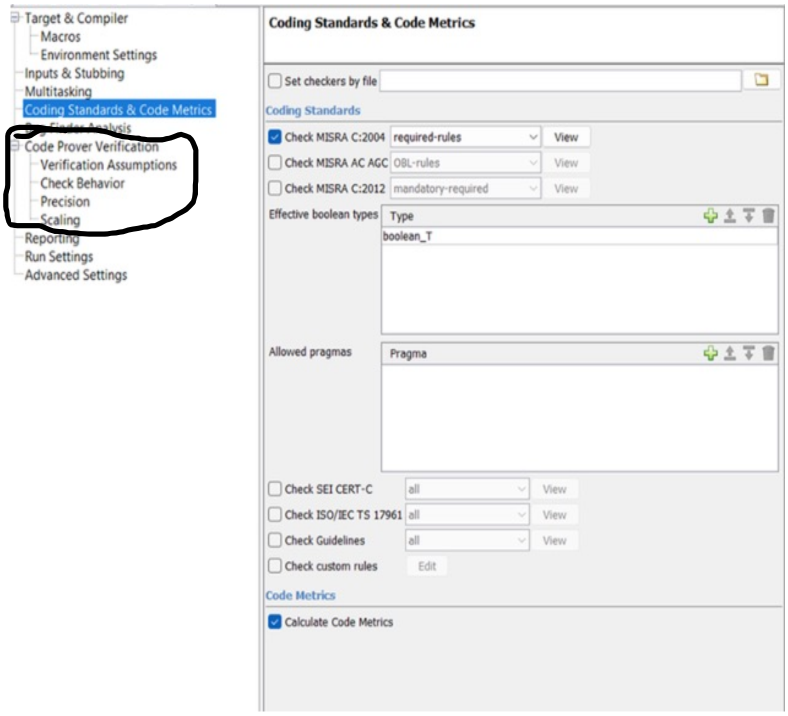
6. Static Code Analsysis (Polyspace)

Bug Finder:

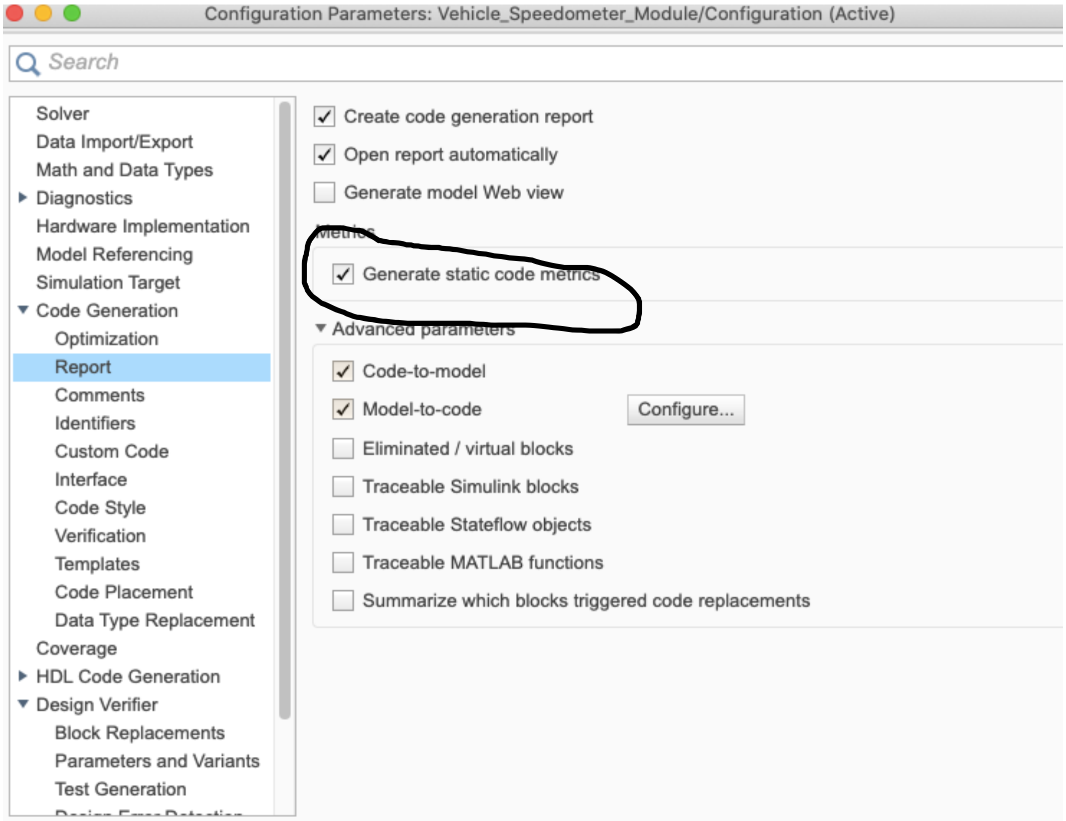


- In the configuration settings option called Target & compiler select the source code language as c, change the target processor as X86_64.

Code Prover:



- In Configuration>Code prover verification tab
- In Precision Select the Precision level -2 and Verification Level is Software Safety Analysis Level-2. then After Click on Save Button the Configuration setting will be save then After one can to go for the Code Prover running.



- For static code ananlysis, one has to check up the <u>*Generate static code metrics*</u> during Code Generation stage.