POLITECNICO DI TORINO

Master's Degree in Computer Engineering - Embedded Systems

Master's Degree Thesis

SystemC-AMS for Modeling heterogeneous systems in automotive domain

Supervisors

Prof. Sara VINCO

Prof. Ernesto SANCHEZ

Candidate

Mohamed SHEHAB

July 2023

Abstract

Model-based software design is a fundamental approach used in automotive industry to develop reliable and efficient systems. This thesis explores the application of Model in the Loop (MIL) technique using SystemC-AMS instead of Simulink, providing valuable insights into the benefits and limitations of employing SystemC-AMS in this context. My reference case study for the modeling and analysis is Semi-active suspension system, which is an automotive safety-critical application designed to improve performance and to guarantee better comfort of the vehicle occupants.

The thesis begins by presenting a comprehensive review of model-based software design principles and the V-cycle methodology, emphasizing their significance in system development. Subsequently, the focus shifts to the utilization of SystemC-AMS as an alternative to Simulink for modeling the Semi-active suspension system. The models developed in SystemC-AMS are compared against their Simulink counterparts, using all fixed-step solvers available in Simulink.

Throughout the thesis, specific attention is given to the limitations of SystemC-AMS, such as the absence of certain components (e.g., a multiple input adder and a discrete time integrator) and the limited support for dynamic systems (that forces the addition of delay blocks, not necessary in Simulink). By acknowledging these limitations, the thesis aims to provide a balanced perspective on the applicability of SystemC-AMS in modeling complex systems, by looking at both accuracy and simulation time.

The comparison between SystemC-AMS and Simulink models is carried out under various scenarios using different input stimuli and different solver settings and obtaining the simulation results. Error estimation will allow to further assess the accuracy of SystemC-AMS w.r.t. Simulink, proving a good level of accuracy of up to 99.99% in several simulation configurations, paired with faster simulation speedup reaches up to 17.26 times compared to Simulink.

In conclusion, this thesis contributes to the understanding of SystemC-AMS as a potential alternative to Simulink for automotive system modeling. By leveraging a reference model and aligning with Simulink modeling, the thesis facilitates a meaningful comparison and provides valuable insights for the automotive industry. The findings will aid in the exploration of more advanced modeling techniques and foster the development of efficient and accurate automotive system simulations using SystemC-AMS.

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

When it comes to complex safety critical systems, such as those found in industries like aerospace, automotive, or healthcare, the need for efficient development and reduced time to market becomes even more paramount. To meet these demands, model based software design has emerged as a necessary approach. By utilizing models to represent system behavior, this methodology enables engineers to streamline the design process, identify potential issues early on, and validate critical functionalities before implementation.With model-based software design, automotive companies can effectively reduce time to market, while ensuring the highest standards of safety, reliability, and efficiency in their products or services. In the automotive industry, model based software design has emerged as a fundamental approach for developing reliable and efficient systems. This approach leverages the power of models to capture system behavior, analyze designs, and ensure the correctness of software implementations. The V-cycle methodology, with its systematic and iterative development process, complements the model-based approach by enabling effective verification and validation of the system design. However, the choice of modeling tools and techniques greatly influences the effectiveness and efficiency of the design process.

This thesis delves into the integration of model based software design and the V-cycle methodology, with a particular emphasis on the application of Model in the Loop (MIL) technique using SystemC-AMS instead of the widely used Simulink. The objective is to explore the feasibility and benefits of employing SystemC-AMS as an alternative modeling tool for automotive systems. The thesis focuses on a specific case study of modeling and analyzing semi-active suspension systems, which serves as a reference for evaluating the capabilities and limitations of SystemC-AMS in this context.

The thesis commences with a comprehensive review of the principles underlying model based software design and the V-cycle methodology. It emphasizes the significance of these approaches in the development of complex automotive systems,

where reliable and efficient software is crucial. By establishing a theoretical foundation, the thesis sets the stage for the subsequent exploration of SystemC-AMS as a potential replacement for Simulink.

A critical aspect of this thesis is the comparison between SystemC-AMS and Simulink models. The models developed in SystemC-AMS are carefully evaluated against their Simulink counterparts, under different scenarios, considering different input stimuli and solver settings. By rigorously assessing the simulation results, the thesis aims to provide an objective analysis of SystemC-AMS in relation to Simulink.

Throughout the thesis, specific attention is given to the limitations and drawbacks of SystemC-AMS. Certain components may be absent in SystemC-AMS compared to Simulink, which could impact its applicability for modeling complex automotive systems. By acknowledging these limitations, the thesis aims to provide a balanced perspective on the strengths and weaknesses of SystemC-AMS and its potential as an alternative modeling tool.

To further assess the accuracy of SystemC-AMS compared to Simulink, the thesis employs analytical techniques, such as error calculations. These techniques allow for a quantitative evaluation of the models' performance in the given conditions, providing a deeper understanding of the differences between the two approaches.

In conclusion, this thesis contributes to the understanding of SystemC-AMS as a potential alternative to Simulink for modeling automotive systems. By employing a reference case study and aligning with Simulink's modeling rules, the thesis facilitates a meaningful and comprehensive comparison between the two tools. The findings and insights gained from this research will aid the automotive industry in exploring more advanced modeling techniques and foster the development of efficient and accurate automotive system simulations using SystemC-AMS.

Chapter 2

Background

2.1 Model Based Software Design

Model based software design involves the creation and utilization of models to represent different aspects of a software system. A model serves as an abstraction that captures the system's behavior, structure, and interactions, enabling analysis, simulation, and validation. The key principles of model based software design include:

- Abstraction: Models provide a simplified representation of the system, focusing on relevant aspects while omitting unnecessary details.
- Separation of Concerns: Models enable the decomposition of complex systems into modular components, each representing a specific aspect or functionality.
- Reusability: Models can be reused across different stages of the software development lifecycle, promoting efficiency and consistency.
- Early Validation: Models facilitate early system verification and validation, enabling identification and resolution of design issues before implementation.

Model based software design offers several advantages over traditional code-centric approaches. These include:

- Improved Understanding: Models provide visual representations that enhance stakeholders' understanding of the system, promoting effective communication and collaboration.
- Iterative Development: Models support iterative refinement and evolution, allowing for incremental development and reducing the impact of design changes.

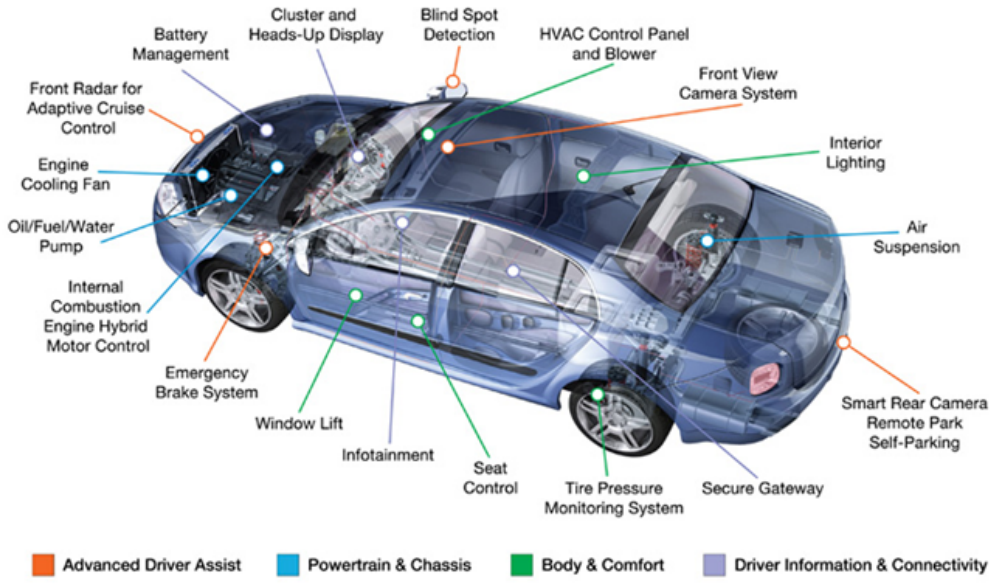

Figure 2.1: Embedded system applications in a vehicle

- Predictability: Models enable the analysis and simulation of system behavior, facilitating performance evaluation, and prediction of system characteristics.
- Testability: Models can be automatically transformed into executable test cases, enabling systematic testing and verification of system requirements.

2.2 V-Cycle Methodology

The V-cycle methodology, also known as the verification and validation cycle, provides a systematic framework for managing the software development process. The V-shape representation of the cycle illustrates the parallel development and testing activities. The key phases of the V-cycle include:

- Requirements Analysis: In this phase, the system requirements are defined, analyzed, and translated into specific software requirements.
- High-Level Design: The high-level design phase focuses on defining the system architecture, modules, interfaces, and the overall structure of the software.
- Detailed Design: This phase involves refining the high-level design by specifying detailed algorithms, data structures, and module interactions.

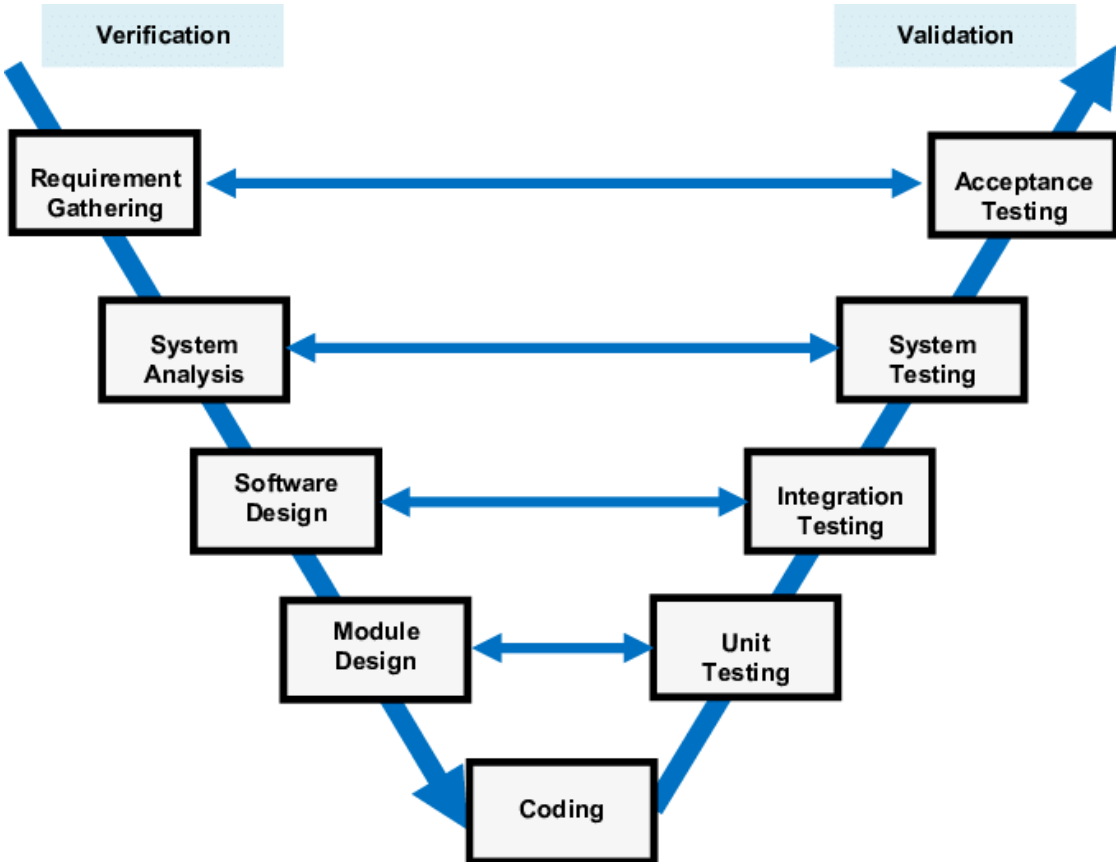

Figure 2.2: V-model life cycle

- Implementation: The implementation phase involves translating the design specifications into executable code.
- Integration and Testing: In this phase, individual software components are integrated and tested as a whole system to ensure their proper functioning and interaction.
- Validation and Verification: The final phase focuses on validating the software against the specified requirements and verifying its correctness and adherence to quality standards.

The V-cycle methodology offers several benefits, including:

- Structured Approach: The V-cycle provides a clear roadmap for the software development process, ensuring that each phase is completed systematically and with proper validation.

- Early Detection of Issues: The methodology emphasizes early verification and validation, enabling the detection and resolution of design and implementation issues in the early stages.

- Traceability: The V-cycle promotes traceability by establishing clear links between requirements, design artifacts, and testing activities, facilitating effective.

2.3 Simulink

Simulink is a graphical tool that is used to model, simulate, and analyze dynamic systems. It provides a block diagram approach where system components, represented by blocks, can be interconnected to represent the behavior of the entire system. It supports a wide range of domains such as control systems, signal processing, communications, image processing, and more. It offers a vast library of pre-built blocks that can be used to construct complex models, and it also allows users to create custom blocks using MATLAB programming.

It is considered the most used tool for MBSD, as it offers code generated from the models, instead of manually writing thousands of lines of code, Embedded Coder is able to automatically generate high quality C, C++,VHDL code, which has the same behavior as the model created in Simulink. It also extends MATLAB Coder and Simulink Coder with advanced optimizations for precise control of the generated functions, data, and files. Therefore, it is possible to perform model in the loop and software in the loop testing efficiently and early.

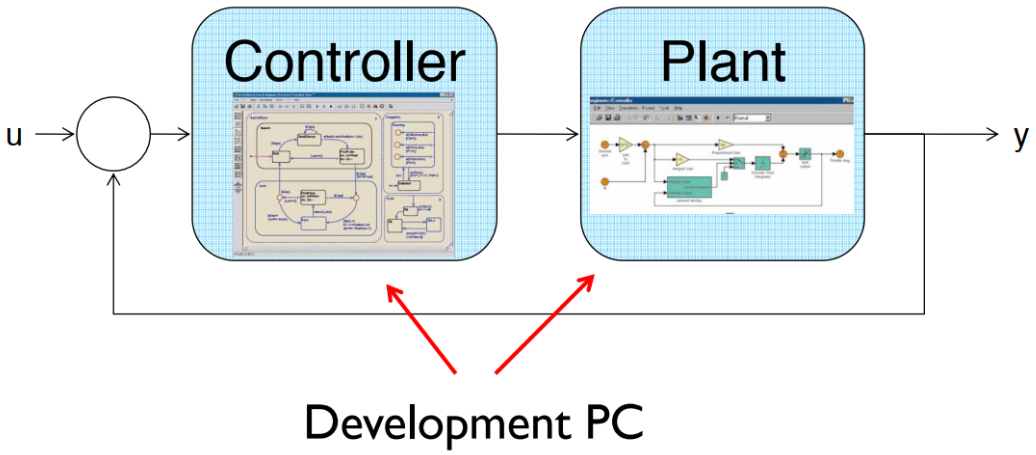

Figure 2.3: Model in the loop testing using Simulink

2.4 SystemC-AMS

SystemC-AMS is an extension of the SystemC language that allows for the modeling and simulation of mixed-signal and analog/mixed-signal (AMS) systems[4]. It provides a framework for designing and simulating complex electronic systems that consist of both digital and analog components.

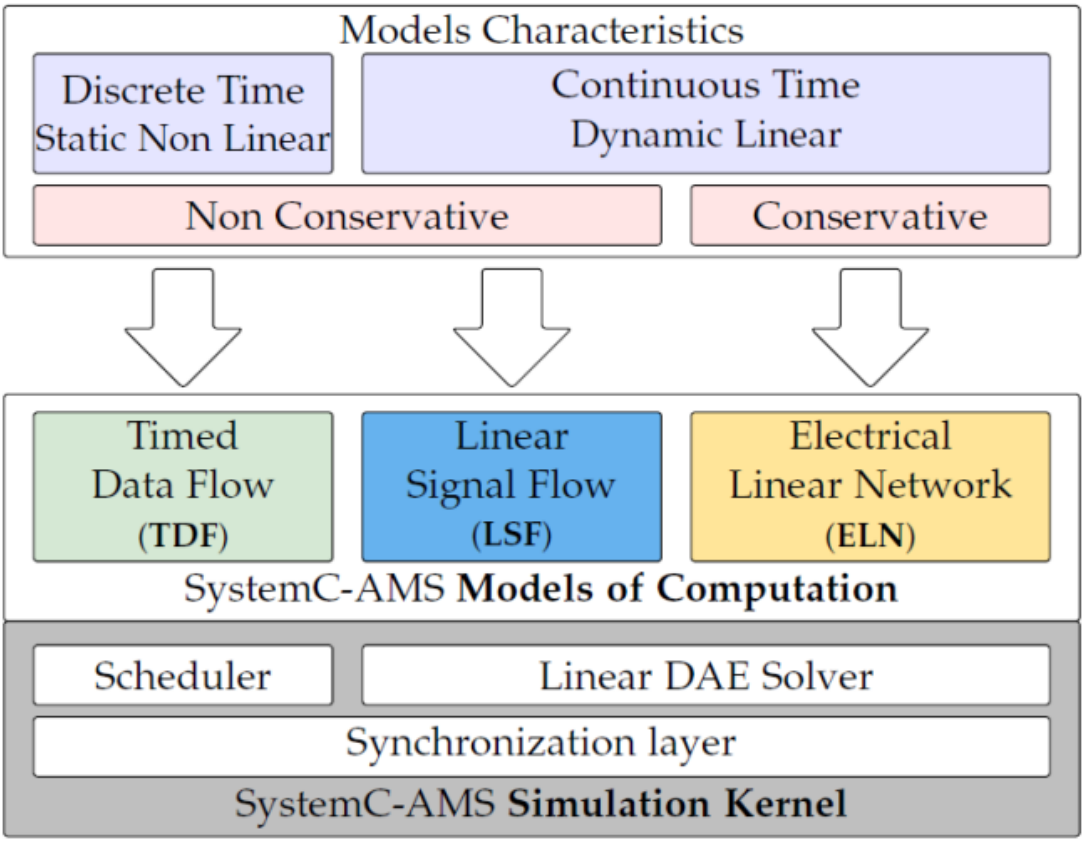

Figure 2.4: SystemC-AMS organization and supported modeling styles

2.4.1 SystemC-AMS use cases and requirements

Here are some common use cases for SystemC-AMS[1]:

- Analog and Mixed-Signal System Modeling: SystemC-AMS enables engineers to model and simulate complex analog and mixed-signal systems, which involve both digital and analog components. It allows for the description of continuous-time behavior, signals, analog/mixed-signal components, and their

interactions.

- System-Level Design and Verification: SystemC-AMS can be used for system-level design and verification of mixed-signal systems. It allows engineers to model the behavior of the complete system, including both digital and analog components, and verify the system's functionality and performance.

- Hardware-Software Co-design: SystemC-AMS is often used for hardware-software co-design, where both digital and analog components are integrated into a single system. It enables the modeling and simulation of the interaction between digital hardware, software, and analog/mixed-signal components.

2.4.2 SystemC-AMS MOCs

In SystemC-AMS, there are three main modeling and simulation domains, each with its own modeling style and level of abstraction: LSF (Linear Signal Flow), TDF (Timed Data Flow), and ELN (Electronic Linear Network). These domains provide different levels of abstraction and are suitable for modeling different aspects of an AMS system.

LSF (Linear Signal Flow) MOC

LSF is the highest level of abstraction in SystemC-AMS and is used for modeling continuous-time analog systems. It provides a natural way to describe systems using linear ordinary differential equations (ODEs) or transfer functions. LSF models consist of interconnected modules that represent analog blocks, such as filters, amplifiers, and sensors. These modules exchange signals, which are continuous-time waveforms, and the behavior of the system is described by differential equations or transfer functions.

LSF models are characterized by their simplicity and ease of use. They are suitable for system-level design and analysis, where high-level performance metrics are of interest, rather than detailed signal-level behavior. LSF models can be used to simulate the overall behavior of an AMS system, but they may not capture the detailed effects of noise, non-linearities, and discrete events.

TDF (Timed Data Flow) MOC

TDF is an intermediate level of abstraction in SystemC-AMS and is used for modeling mixed-signal systems that involve both continuous-time and discrete-time components. TDF models represent systems using discrete-event modeling

techniques, where events occur at discrete points in time. TDF models are composed of modules that exchange data samples, which are discrete-time values.

TDF models can describe both analog and digital behaviors. Analog behaviors are typically modeled using LSF-style constructs, while digital behaviors are modeled using discrete-event modeling techniques similar to those used in SystemC. TDF models are well-suited for modeling mixed-signal systems that involve digital signal processing (DSP) algorithms, control systems, or other discrete-time components.

TDF models provide a balance between the simplicity of LSF models and the detailed signal-level behavior captured by ELN models. They are suitable for system-level exploration, architectural analysis, and performance estimation, as well as for verifying the interaction between analog and digital components.

ELN (Electronic Linear Network) MOC

ELN (Electronic Linear Network) ELN is the lowest level of abstraction in SystemC-AMS and is used for modeling the detailed electrical behavior of analog systems. ELN models represent systems using a network of linear electrical components, such as resistors, capacitors, and inductors, interconnected by voltage and current sources. ELN models capture the behavior of electrical circuits and systems at the transistor level.

ELN models provide a high level of accuracy and capture the detailed signal-level behavior of analog circuits. They are suitable for analyzing the electrical characteristics of individual components, designing analog integrated circuits (ICs), and performing detailed analog circuit simulations.

ELN models can be used in combination with LSF or TDF models to create hybrid models that capture both the high-level system behavior and the detailed electrical characteristics of analog components. This allows for a multi-level modeling approach, where different levels of abstraction are used to model different parts of the system based on their level of complexity and the desired level of accuracy. In summary, ranging from high-level system exploration to detailed analog circuit analysis.

Chapter 3

The reference case study

3.1 The semi-active suspension system

The semi-active suspension system is an automotive safety-critical application designed to improve performance and to guarantee better comfort of the vehicle occupants. It is adopted where a control low is used to change the damping coefficient in relation to the suspended mass and wheel speed[7]. Thanks to a mechatronic system based on electro-valves, the Skyhook controller changes the damping coefficient of each of the 4 suspensions installed in the car to impose a higher damping coefficient when the suspension itself is expanding and a lower one when it is compressing. This allows to maintain the maximum possible stability for the vehicle body regardless of driving and road conditions: the system damps the vibrating body in comparison to an imaginary line in the horizon. The semi-active suspension system consists of two main parts: quarter car model and Skyhook controller.

3.1.1 Quarter car model

The quarter car model is a simplified mathematical model used to describe and analyze the vertical dynamic behavior of a vehicle's suspension system[2][6]. It is called a "quarter car" model as it represents one-fourth of the entire vehicle, focusing on the interaction between a single wheel and its associated suspension components. The dynamic equation describing the suspension system is the following:

$$\begin{bmatrix} m_1 & 0 \\ 0 & m_2 \end{bmatrix} \begin{bmatrix} \ddot{x_1} \\ \ddot{x_2} \end{bmatrix} + \begin{bmatrix} c & -c \\ -c & c \end{bmatrix} \begin{bmatrix} \dot{x_1} \\ \dot{x_2} \end{bmatrix} + \begin{bmatrix} k_1 & -k_1 \\ -k_1 & k_1 + k_2 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 0 \\ k_2 \end{bmatrix} \begin{bmatrix} x_{in} \end{bmatrix} \tag{3.1}$$

Where $\dot{x}_1$, $\dot{x}_2$, $\ddot{x}_1$, and $\ddot{x}_2$ represent, respectively, the masses' velocities and accelerations. Blocks m1 and m2 represent, respectively, the car and wheel masses involved

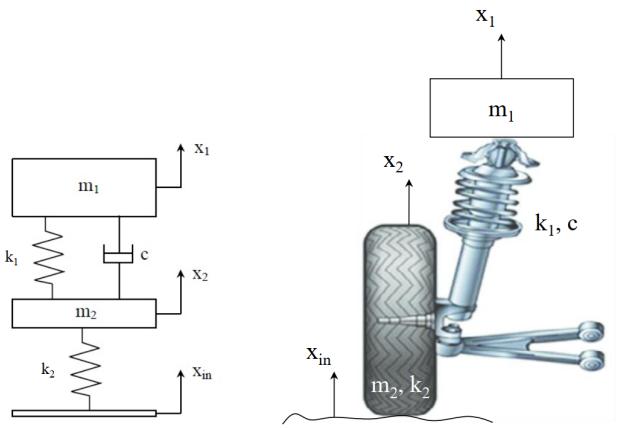

Figure 3.1: The quarter car model

in the suspension system, while springs k1 and k2 are the car and wheel masses stiffness. The quantities x1 and x2 represent, respectively, the displacements of the car and wheel masses (assuming that only a vertical motion is allowed), and the vertical direction is conventionally considered positive; x_{in} is the displacement input acting on the wheel, modelling road profile. The damper c is modelling the damping action of the suspension system which only consists of a spring (k1, in this case) and the damper itself. In case of a passive system, the value of c is a constant parameter set by the manufacturer and cannot be modified. In case of semi-active systems, c can be modified while driving.

The previous equation can be split into two separate equations, yielding the vertical accelerations for the two masses:

$$\ddot{x}_1 = (k_1 x_2 - k_1 x_1 - c\dot{x}_1 + c\dot{x}_2)\frac{1}{m_1} \tag{3.2}$$

$$\ddot{x}_2 = (k_2 x_{in} + c\dot{x}_1 - c\dot{x}_2 + k_1 x_1 - (k_1 + k_2)x_2)\frac{1}{m_2} \tag{3.3}$$

Typical values for parameters k1, k2, m1 and m2 that are used for the simulation are the following:

PARAMETER	VALUE
Car body mass (m1)	380Kg
Wheel body mass (m2)	31Kg
Suspension stiffness (k1)	29.000N/m
Wheel stiffness (k2)	228.000N/m
Damping (c)	1.500Ns/m

Table 3.1: Values of the coefficients and masses

3.1.2 The Skyhook controller

Based on road disturbances and motion information, the Skyhook controller calculates the desired damping coefficient for each damper in real time. The Skyhook principle is the following:

$$(\dot{x}_1 - \dot{x}_2) \geq 0 \Rightarrow c = 6000\frac{Ns}{m} \tag{3.4}$$

$$(\dot{x}_1 - \dot{x}_2) \leq 0 \Rightarrow c = 150\frac{Ns}{m} \tag{3.5}$$

3.2 The suspension system in Simulink

3.2.1 The suspension system

The suspension system consists of three main parts: the plant, the mechanical system, which is in our case the quarter car model, and the controller: that contains the Skyhook algorithm. Then we have also in the top left the input stimuli. The harness can be tested in two different modes, first when we have a constant damping coefficient and the second by applying the Skyhook controller.

3.2.2 The plant

The plant consists of four partitions, including three inputs, three gain blocks, two product blocks, two integrators and three outputs. The adder adds the second and fourth inputs and subtracts the others.

3.2.3 The wheel acceleration equation

The wheel acceleration equation consists of five partitions, includes four inputs, four gain blocks, two product blocks, two integrators and three outputs. The adder

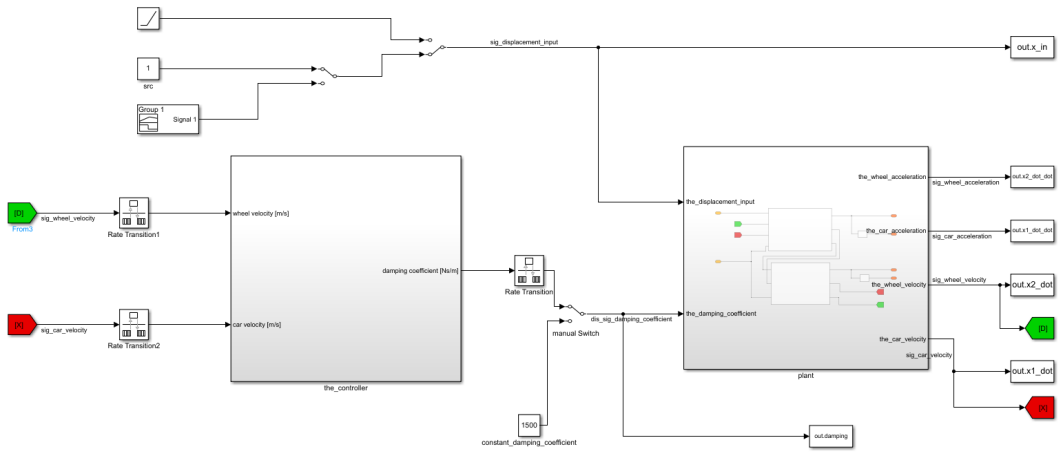

Figure 3.2: The semi-active suspension system in Simulink

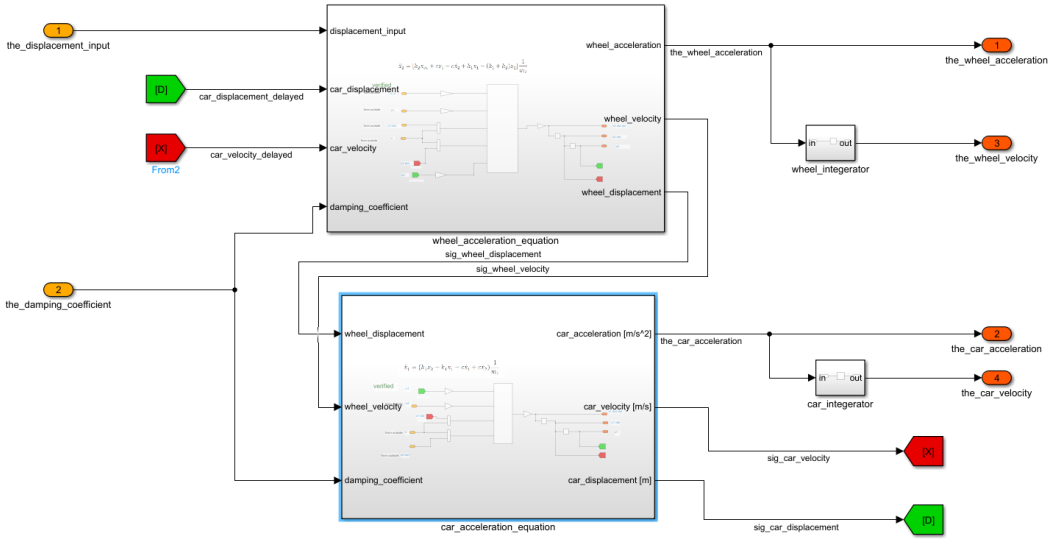

Figure 3.3: The plant, internal layer in Simulink

adds the first three partitions and subtract the last two ones.

3.2.4 The car acceleration equation

In order to maintain the max readability and since we have two dependent dynamic equations, it is better to model each of them separately. wheel acceleration equation in the top and car acceleration equation in the bottom.

$$\ddot{x}_2 = [k_2 x_{in} + c\dot{x}_1 - c\dot{x}_2 + k_1 x_1 - (k_1 + k_2)x_2]\frac{1}{m_2}$$

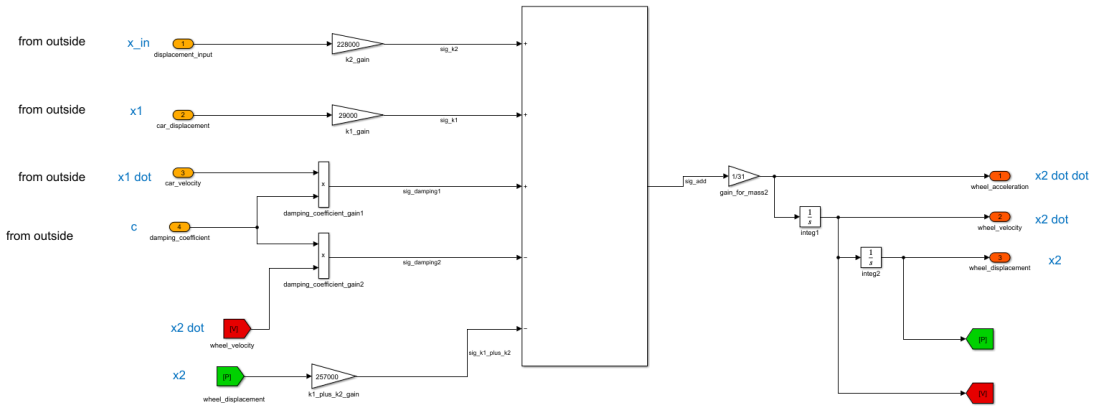

Figure 3.4: The wheel acceleration equation in Simulink

$$\ddot{x}_1 = (k_1 x_2 - k_1 x_1 - c\dot{x}_1 + c\dot{x}_2)\frac{1}{m_1}$$

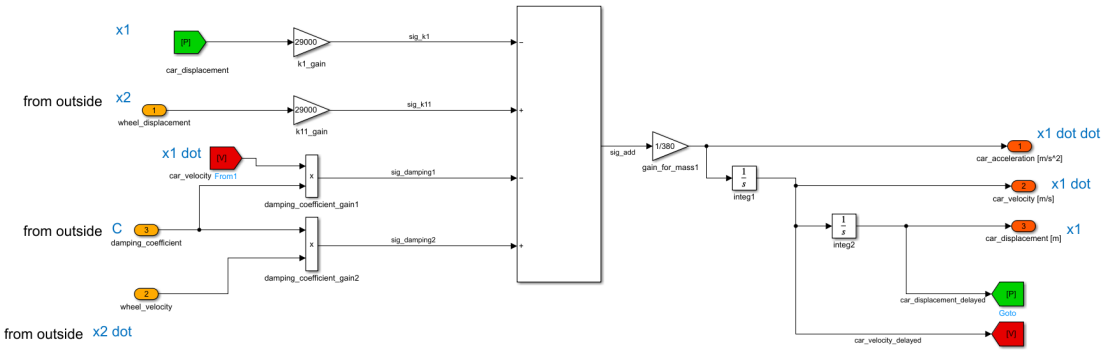

Figure 3.5: The car acceleration equation in Simulink

3.2.5 The Skyhook controller

The implementation of Skyhook controller is very simple. It takes the wheel and car velocities from the plant and subtract them, then multiplies by the car velocity. Finally, it compares the result, if greater than zero, then it gives the maximum damping coefficient, else the minimum.

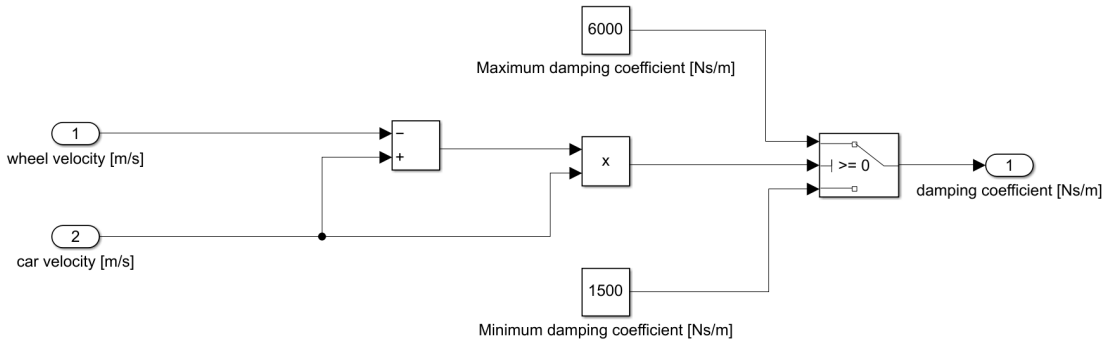

Figure 3.6: The skyhook controller in Simulink

3.2.6 Simulink configuration parameters

In order to select the solver settings according to the requirement the controller should work in 1 kHz, the solver should be fixed step type for the controller, for the plant could work with a fixed or variable step solver, in our case it is a fixed step solver ode4 (Runge-Kutta) that is suitable for for dynamic systems.

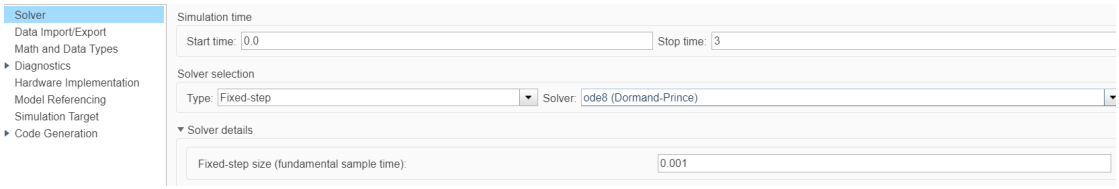

Figure 3.7: Simulink solver settings

Chapter 4

Modeling in SystemC-AMS

4.1 The procedures for modeling in SystemC-AMS

The procedures I followed for modeling in SystemC-AMS during the thesis are the following:

1. Adapting and using Simulink to capture the SystemC-AMS model taking into account the difference between the components in both of them.

2. Writing the C++ code for each model in visual studio, compiling it and generating the .dat file that contains the simulation results.

3. Comparing the results obtained from Simulink with the results of SystemC-AMS in Matlab.

4. Analysing and simulating the results of both of them.

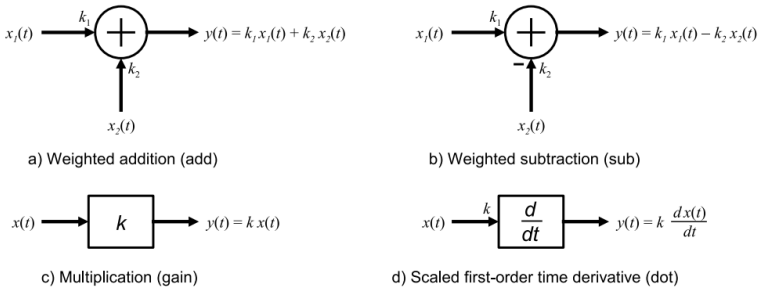

Figure 4.1: Some basic LSF primitives

4.2 Modeling the plant

First thing I tried to model the plant and test it. As mentioned in chapter one, SystemC-AMS has three MOCs, and the most suitable one to model the continuous plant is LSF. It has a finite set of predefined LSF primitive modules implementing functions such as addition, multiplication, integration, etc,[5] as shown in Figure 4.1. Since we will have to add the Skyhook controller, which will be implemented in TDF, there is no issue because LSF has also a converter module called Source that read from a TDF signal and write to an LSF signal., and another module called Sink that read an LSF signal and write the equivalent values to a TDF signal. The plant in our case is the system that describe the two dynamic equations: wheel acceleration equation and car acceleration equation. This is our main plant that will be tested and simulated before and after adding the Skyhook controller.

4.2.1 First model

In order to model a system there are three basic elements: first the main model, in our case is the plant, then the input stimuli and tracing the output. For the plant I combined both of the two basic equations in one model, trying to minimize the components used for modeling.

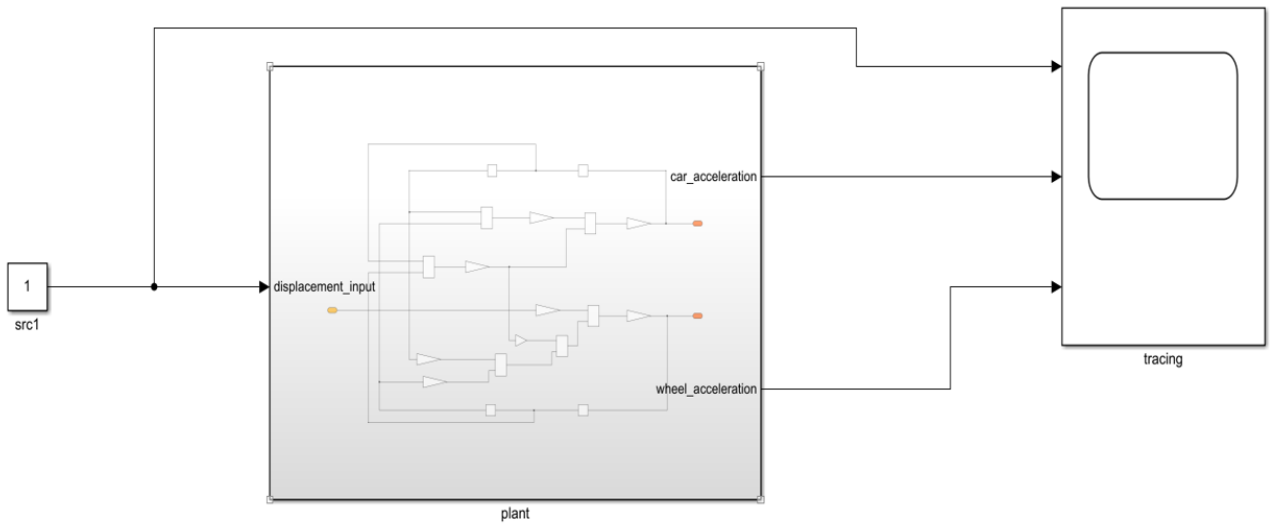

Figure 4.2: The top layer of first model in SystemC-AMS

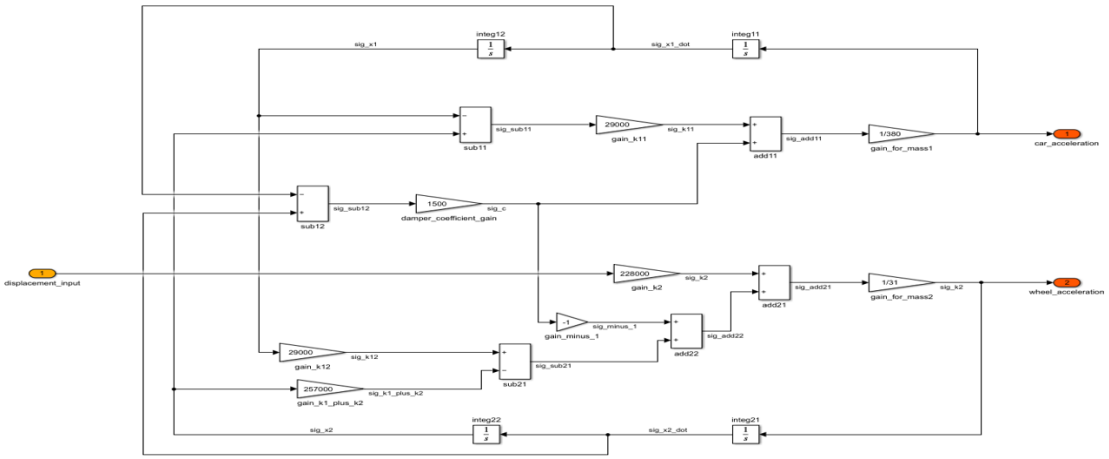

Figure 4.3: The plant of first model in SystemC-AMS

SystemC-AMS code for first model

It consists of two modules besides the top layer which is the plant, as shown in Listing 4.1.

Listing 4.1: the plant of first model in c++

```
1  //================================================
2  //plant.cpp
3  //================================================
4
5  #include <systemc.h>
6  #include <systemc-ams>
7  #include "quarter_car_model.h"
8  #include "signal_builder.h"
9
10 int sc_main(int argc, char* argv[]) {
11
12   sc_core::sc_set_time_resolution(1.0, sc_core::SC_FS);
13
14   sca_lsf::sca_signal sig_src,the_displacement_input,
      the_car_acceleration, the_wheel_acceleration;
15
16   signal_builder my_signal_builder("signal_builder");
17   my_signal_builder.constant_input(sig_src);
18   my_signal_builder.ramp_output(the_displacement_input);
19
20   quarter_car_model my_quarter_car_model("
      my_quarter_car_model", 1500.0, 1.0 / 380.0, 1.0 / 31.0,
21      29000.0, 29000.0, 228000.0, -1, 257000.0);
```

```
22    my_quarter_car_model.displacement_input(
       the_displacement_input);
23    my_quarter_car_model.car_acceleration(the_car_acceleration
       );
24    my_quarter_car_model.wheel_acceleration(
       the_wheel_acceleration);
25
26    return 0;
27  }
28
29  //=====================================================
```

Then we have first module which is the quarter car model. It includes gain blocks, integrator blocks, and arithmetic blocks to model the equations. The module has one input(the displacement input) and two output ports: wheel acceleration and car acceleration. It has also 8 gain blocks, four integer blocks (two for first equation and two for the second), three add blocks (one for first equation and two for the second) and three sub block (two for first equation and one for the second).

Listing 4.2: the quarter car module of first model in c++

```
1  //=================================================
2  //quarter_car_model.h
3  //=================================================
4  SC_MODULE(quarter_car_model)
5  {
6  sca_lsf::sca_in displacement_input;
7  sca_lsf::sca_out car_acceleration;
8  sca_lsf::sca_out wheel_acceleration;
9
10 sca_lsf::sca_gain damper_coefficient_gain,
11 ...
12 gain_k1_plus_k2;
13
14 sca_lsf::sca_integ integ11 ... integ22;
15
16 sca_lsf::sca_add add11 ... add22;
17
18 sca_lsf::sca_sub sub11 ... sub21;
19
20
21 quarter_car_model(sc_core::sc_module_name nm, double
     damping_coefficient ... double k1_plus_k2);
22
23 private:
24   sca_lsf::sca_signal sig_x1_dot, sig_x2_dot,
```

```
25  ...
26      sig_add11;
27  }
28
29  //=====================================================
```

Listing 4.3: the quarter car module of first model in c++

```
1  //=================================================
2  //quarter_car_model.cpp
3  //=================================================
4
5  quarter_car_model::quarter_car_model(sc_core::sc_module_name
       nm, double damping_coefficient...double k1_plus_k2)
6  : displacement_input("displacement_input"), car_acceleration
     ("car_acceleration"), wheel_acceleration("
     wheel_acceleration"),
7
8  damper_coefficient_gain("damper_coefficient_gain",
     damping_coefficient),
9    gain_for_mass1("gain_for_mass1",one_over_mass_1),
10 ...
11   integ11("integ11")...integ22("integ22"),
12   add11("add11")...sub21("sub21"),
13 ...
14 sig_x1_dot("sig_x1_dot")...sig_x2_dot("sig_x2_dot")
15 {
16   gain_for_mass1.x(sig_add11);
17   gain_for_mass1.y(car_acceleration);
18 ...
19   sub21.x1(sig_k12);
20   sub21.x2(sig_k1_plus_k2);
21   sub21.y(sig_sub21);
22 }
23 //=====================================================
```

Now we have the plant ready, in order to provide the test stimuli for lsf we have to use the lsf source. It is a module that generates a continuous-time signal. It can be used to model various signal sources in an analog system, such as voltage or current sources. In our case it is used as continuous-time signal source that represent the displacement input or that is part of the system being modeled.

Listing 4.4: The lsf source

```
1  //=================================================
2    //the source for the displacement input:
3    double init_value = 0.0;
```

```
4     double offset = 1;
5     double amplitude = 0.0;
6     double frequency = 0.0;
7     double phase = 0.0;
8     sca_core::sca_time delay = sca_core::sca_time(1, sc_core::
       SC_MS);
9
10    sca_lsf::sca_source src("src"
11      , init_value, offset, amplitude,
12      frequency, phase, delay); // step of 1 unit at t=1ms
13    src.y(sig_src);
14    src.set_timestep(1, sc_core::SC_MS);
15
16  //=====================================================
```

Although LSF source can be used to generate several continuous signals, it cannot provide a ramp signal, which I need for simulating, therefore I created another module, which is signal builder. It receives constant signal and produce the ramp signal. It contains mainly an integer block.

Listing 4.5: the signal builder module of first model in c++

```
1  //=================================================
2  //signal_builder_model.h
3  //=================================================
4  SC_MODULE(signal_builder) //Here is our plant(the quarter
       car model),we can describe it with two equations.
5  {
6
7    sca_lsf::sca_in constant_input;  //the constant_input
8
9    sca_lsf::sca_out ramp_output;   //the ramp_output
10
11   sca_lsf::sca_integ integ1;
12
13   signal_builder(sc_core::sc_module_name nm);
14
15 };
16 //=====================================================
```

Listing 4.6: the signal builder module of first model in c++

```
1  //=================================================
2  //signal_builder_model.cpp
3  //=================================================
4  signal_builder::signal_builder(sc_core::sc_module_name nm)
5
```

```
6     :constant_input("constant_input"), ramp_output("
       ramp_output"),
7     integ1("integ1")
8   {
9     integ1.x(constant_input);
10    integ1.y(ramp_output);
11
12  }
13  //=======================================================
```

In order to capture and analyzing the behaviour of signals during simulation we have to call a function supported by SystemC-AMS that create the tracing .dat file that contains the desired input and output signals.

Listing 4.7: Tracing to a tabular file

```
1   //=======================================================
2     sca_util::sca_trace_file* atf = sca_util::
       sca_create_tabular_trace_file("car_wheel_acceleration");
3     sca_util::sca_trace(atf, the_displacement_input, "
       the_displacement_input");
4     sca_util::sca_trace(atf, the_wheel_acceleration, "
       the_wheel_acceleration");
5     sca_util::sca_trace(atf, the_car_acceleration, "
       the_car_acceleration");
6
7     sc_core::sc_start(3.0, sc_core::SC_SEC);
8     std::cout << "Simulation finished." << std::endl;
9
10    sca_util::sca_close_tabular_trace_file(atf);
11  //=======================================================
```

4.2.2 Second model

Although first model functionally works, it is not well organized model, for this reason I made the second model to be similar to the model of Simulink. The good thing that all the changes I needed was in the quarter car module. It contains of wheel and car equation describing each of them separately but in the same module.

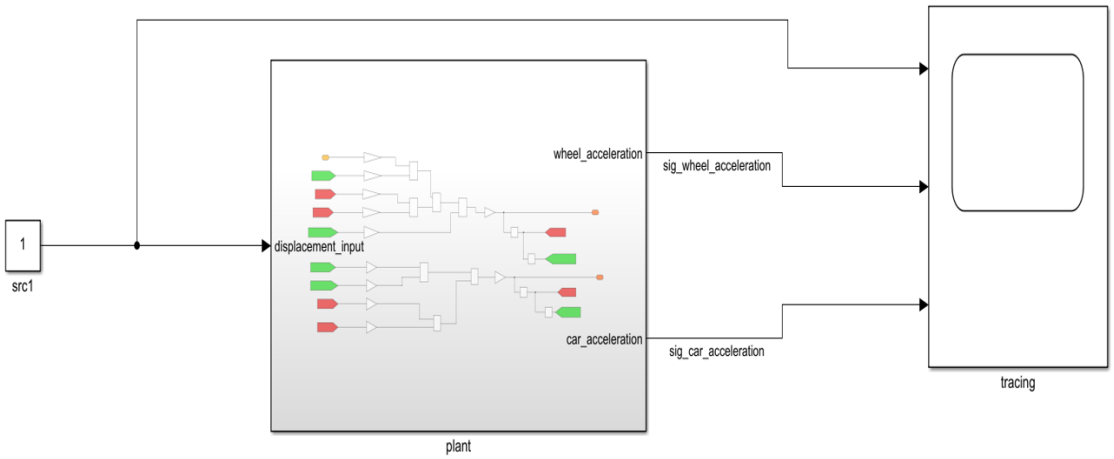

Figure 4.4: Second model of the plant, top layer in SystemC-AMS

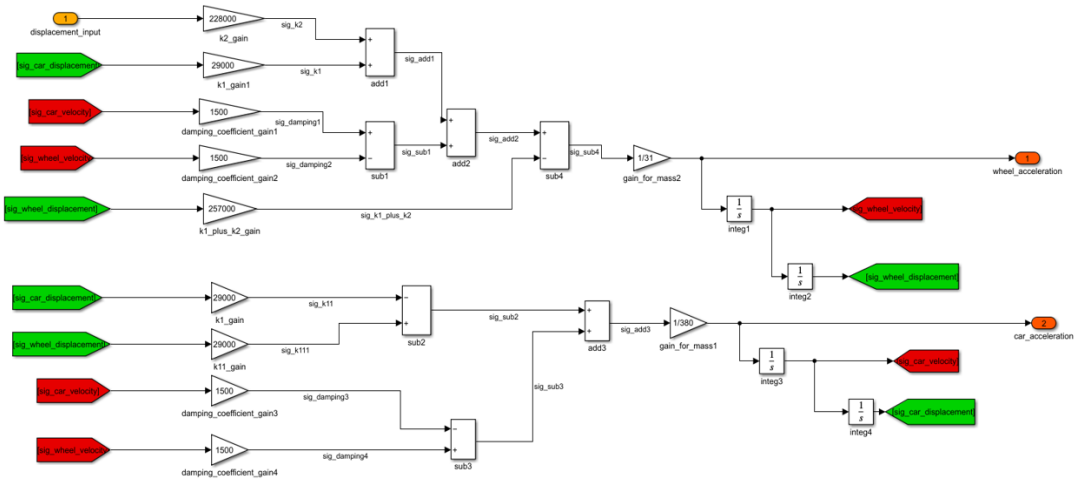

Figure 4.5: Second model of the plant, internal layer in SystemC-AMS

SystemC-AMS code for second model

Due to the modality, I had not to write the whole code from scratch, I needed only to replace the quarter car module.

Listing 4.8: the quarter car module of second model in c++

```
1 //================================================
2 //quarter_car_model.h
3 //================================================
```

```
4  SC_MODULE(quarter_car_model) //Here is our plant(the quarter
       car model),we can describe it with two equations.
5  {
6    sca_lsf::sca_in displacement_input;
7    sca_lsf::sca_out wheel_acceleration;
8    sca_lsf::sca_out car_acceleration;
9
10   sca_lsf::sca_gain k2_gain...gain_for_mass1;
11
12   sca_lsf::sca_add add1, add2, add3;
13
14   sca_lsf::sca_sub sub1, sub2, sub3, sub4;
15
16   sca_lsf::sca_integ integ1...integ4;
17
18   quarter_car_model(sc_core::sc_module_name nm);
19
20 private:
21   sca_lsf::sca_signal sig_k2, sig_k1,
22  ...
23     sig_car_displacement;
24 };
25 //=====================================================
```

Listing 4.9: the quarter car module of first model in c++

```
1  //=====================================================
2  //quarter_car_model.cpp
3  //=================================================
4  quarter_car_model::quarter_car_model(sc_core::sc_module_name
       nm)
5  : displacement_input("displacement_input"),
     wheel_acceleration("wheel_acceleration"),
     car_acceleration("car_acceleration"),
6
7  k2_gain("k2_gain",228000.0), k1_gain1("k1_gain1",29000.0),
8  ...
9  integ1("integ1"), integ2("integ2"), integ3("integ3"), integ4
     ("integ4"),
10 ...
11 sig_sub4("sig_sub4"), sig_wheel_velocity("sig_wheel_velocity
     "),
12 ...
13 sig_car_displacement("sig_car_displacement")
14
15 {
```

```
16    k2_gain.x(displacement_input);
17    k2_gain.y(sig_k2);
18
19    k1_gain1.x(sig_car_displacement);
20    k1_gain1.y(sig_k1);
21  ...
22    integ4.x(sig_car_velocity);
23    integ4.y(sig_car_displacement);
24  }
25
26  //=======================================================
```

4.2.3 Third model

This model is similar to the second one but I separated it into several modules. For example instead of having only the quarter car module, it consists of six modules inside the plant, module for the wheel acceleration equation, module for car acceleration equation, and so on. It is better for testing to have several modules that contains small portion of code rather than having only one huge module.

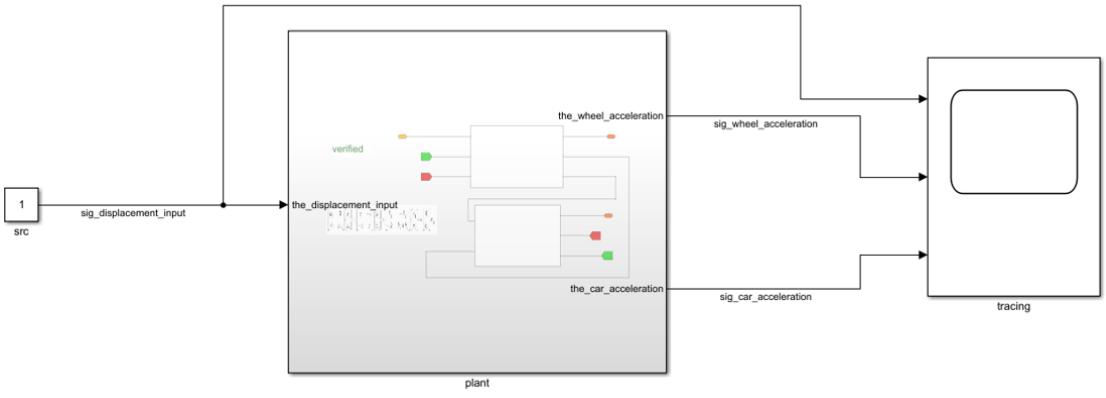

Figure 4.6: Third model of the plant, top layer in SystemC-AMS

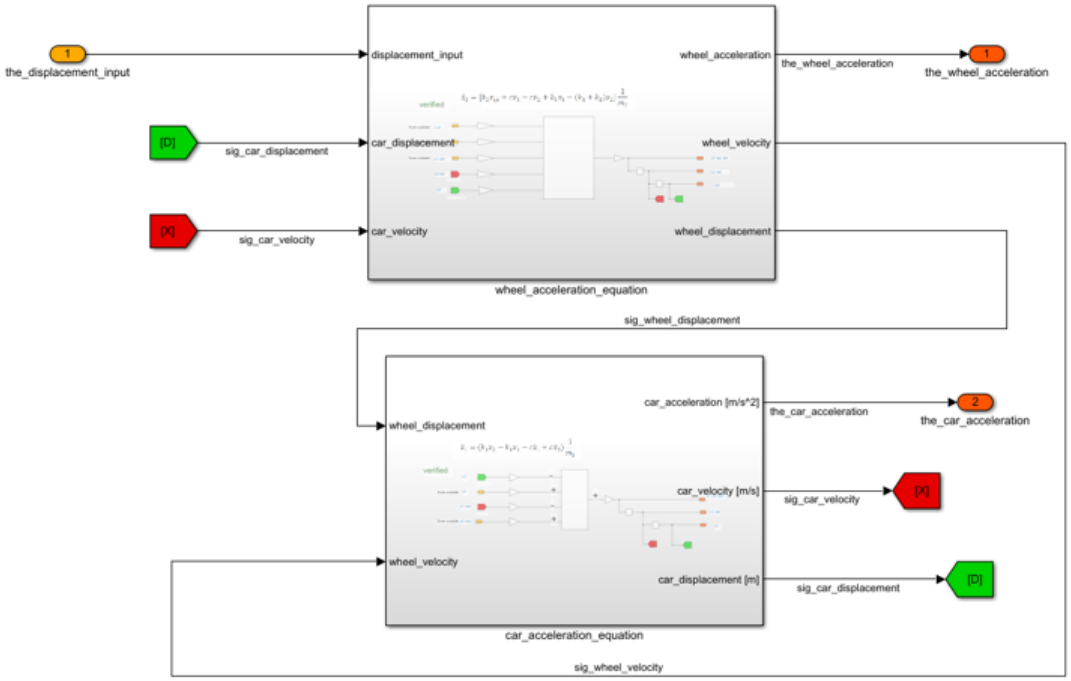

Figure 4.7: Third model of the plant, internal layer in SystemC-AMS

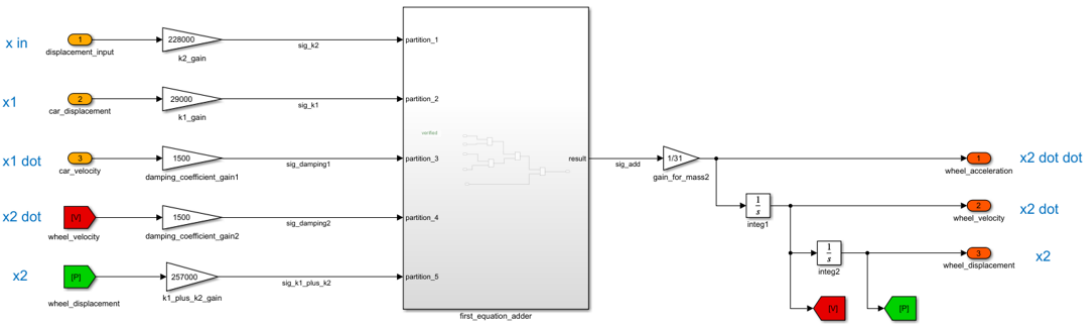

Figure 4.8: Third model of the wheel acceleration equation in SystemC-AMS

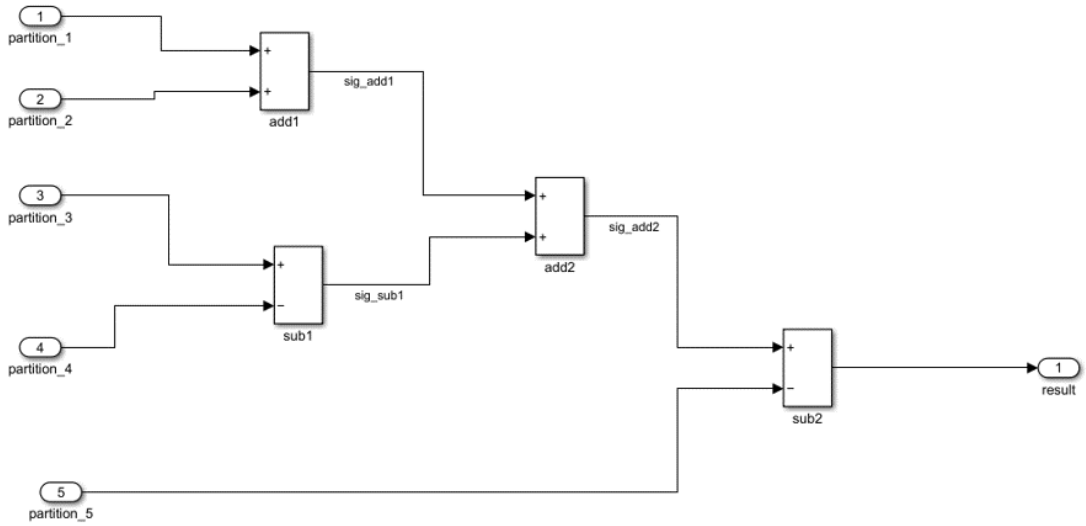

Figure 4.9: Third model, the adder model used for wheel acceleration equation in SystemC-AMS

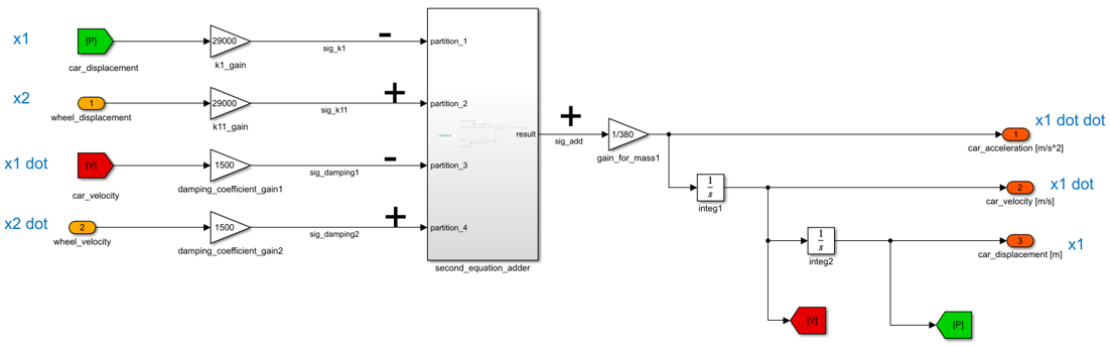

Figure 4.10: Third model of the car acceleration equation in SystemC-AMS

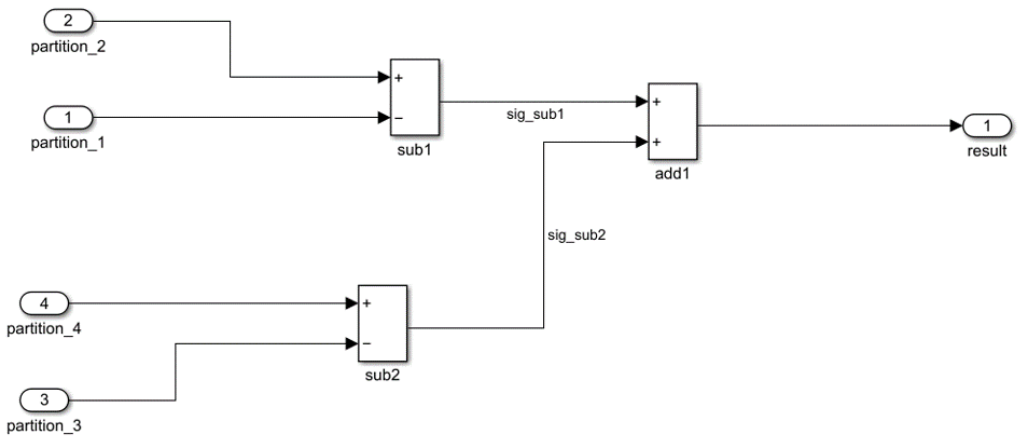

Figure 4.11: Third model, the adder model used for car acceleration equation in SystemC-AMS

SystemC-AMS code for third model

The LSF primitives such as weighted addition and weighted subtraction do not support multiple inputs, for this reason I made a module that could be adapted to perform several additions and subtractions in one module. Similarly for car acceleration equation.

Listing 4.10: first adder for wheel acceleration equation equation

```
1  //==================================================
2  //first_equation_adder.h
3  //==================================================
4  SC_MODULE(first_equation_adder) //the adder of the wheel,it
       behaves like:+++--
5  {
6  sca_lsf::sca_in partition_1;
7  ...
8  sca_lsf::sca_in partition_5;
9  sca_lsf::sca_out result;
10
11 sca_lsf::sca_add add1, add2;
12 sca_lsf::sca_sub sub1, sub2;
13 first_equation_adder(sc_core::sc_module_name nm);
14
15 private:
16   sca_lsf::sca_signal sig_add1, sig_sub1, sig_add2;
17 };
```

```
18 //=====================================================
```

Listing 4.11: first_adder_equation.cpp

```
1  //=====================================================
2  //first_equation_adder.cpp
3  //=====================================================
4  first_equation_adder::first_equation_adder(sc_core::
       sc_module_name nm)
5    : partition_1("partition_1"),
6   ...
7   partition_5("partition_5"), result("result"),
8
9    add1("add1"), sub1("sub1"),
10   add2("add2"), sub2("sub2"),
11
12   sig_add1("sig_add1"), sig_sub1("sig_sub1"), sig_add2("
       sig_add2")
13
14 {
15   add1.x1(partition_1);
16   add1.x2(partition_2);
17   add1.y(sig_add1);
18 ...
19   sub2.x1(sig_add2);
20   sub2.x2(partition_5);
21   sub2.y(result);
22 }
23 //=====================================================
```

Listing 4.12: wheel_acceleration_equation.h

```
1  //=====================================================
2  //wheel_acceleration_equation.h
3  //=====================================================
4  SC_MODULE(wheel_acceleration_equation)
5  {
6  sca_lsf::sca_in displacement_input;
7  sca_lsf::sca_in car_displacement;
8  ...
9  sca_lsf::sca_out wheel_displacement;
10
11 sca_lsf::sca_gain k2_gain ... gain_for_mass2;
12 sca_lsf::sca_integ integ1, integ2;
13
14 first_equation_adder my_first_equation_adder;
```

```
15
16 wheel_acceleration_equation(sc_core::sc_module_name nm);
17
18 private:  sca_lsf::sca_signal sig_k2 ...
19                      sig_add;
20 };
21 //=====================================================
```

Listing 4.13: Third module:wheel_acceleration equation.cpp

```
1  //=================================================
2  //wheel_acceleration_equation.cpp
3  //=================================================
4  wheel_acceleration_equation::wheel_acceleration_equation(
      sc_core::sc_module_name nm)
5    : displacement_input("displacement_input")
6   ...
7    wheel_displacement("wheel_displacement")
8  ...
9  k2_gain("k2_gain",228000.0),
10 ...
11   integ1("integ1"), integ2("integ2"),
12   my_first_equation_adder("my_first_equation_adder"),
13
14   sig_k2("sig_k2")...sig_add("sig_add")
15 {
16   k2_gain.x(displacement_input);
17   k2_gain.y(sig_k2);
18 ...
19   damping_coefficient_gain2.x(wheel_velocity);
20   damping_coefficient_gain2.y(sig_damping2);
21 ...
22   integ2.x(wheel_velocity);
23   integ2.y(wheel_displacement);
24 }
25 //=====================================================
```

Then we have the plant that includes both of the two equations and connect them together in one module.

Listing 4.14: Third module:plant.h

```
1  //=================================================
2  //plant.h
3  //=================================================
4  SC_MODULE(plant)  // here is our plant that include the car
      and wheel equations
```

```
{
  sca_lsf::sca_in the_displacement_input;
 ...
  sca_lsf::sca_out the_car_acceleration;

wheel_acceleration_equation my_wheel_acceleration_equation;
car_acceleration_equation my_car_acceleration_equation;

  plant(sc_core::sc_module_name nm);

private:
  sca_lsf::sca_signal  sig_car_displacement...
   sig_wheel_displacement;
};
//=====================================================
```

Listing 4.15: Third model: plant.cpp

```
//=================================================
//plant.cpp
//=================================================
plant::plant(sc_core::sc_module_name nm)
  : the_displacement_input("the_displacement_input"),
 ...
  the_car_acceleration("the_car_acceleration"),

my_wheel_acceleration_equation("
   my_wheel_acceleration_equation"),
my_car_acceleration_equation("my_car_acceleration_equation")
...
sig_wheel_displacement("sig_wheel_displacement")
{

my_wheel_acceleration_equation.displacement_input(
   the_displacement_input);
...
my_wheel_acceleration_equation.wheel_acceleration(
   the_wheel_acceleration);
my_car_acceleration_equation.wheel_displacement(
   sig_wheel_displacement);
...
my_car_acceleration_equation.car_acceleration(
   the_car_acceleration);
}
//=================================================
```

Then the other modules are the same as in first and second models.

4.3 Modeling the system in SystemC-AMS

Now we have modeled the continuous plant. The third model is the most organised model, and the plant functionally behaves correctly. The system consists of the plant besides the Skyhook controller, therefore we need to model it. In order to model the Skyhook controller it is not suitable to model it using LSF as it is a discrete system. Also the test stimuli should be discrete. The continuous plant need modification in order to interact with the discrete system. Let's start with the modifications of the plant.

4.3.1 Modifications of the plant

I added another input to the plant which is the damping coefficient input. Also I added more four outputs, the car and wheel velocities and displacements in order to trace them later.

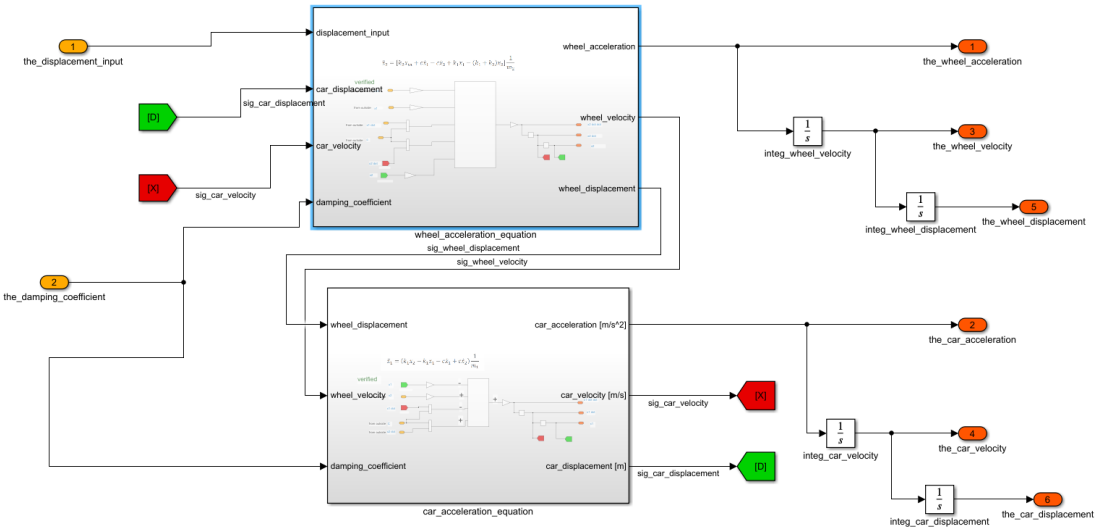

Figure 4.12: The plant after the modifications in SystemC-AMS

The damping coefficient input comes from discrete environment, therefore I used another LSF primitive, which is (sca_tdf::sca_in <double>). Similarly this modification should be also applied in car and wheel acceleration modules. Since there is an interaction between LSF and TDF models, and in order to make the model well and clearly described , it is better to isolate both the continuous plant described in LSF and the whole system described in TDF, for this reason I created

another model named the discrete plant, it includes the continuous plant beside the LSF converter modules. There are two types of LSF converters, from TDF to LSF and from LSF to TDF. I used the first one before the inputs to the continuous plant and the second for the outputs as shown in the following figure:

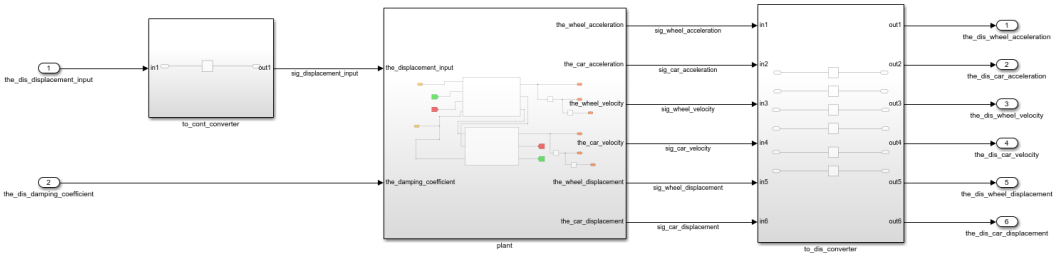

Figure 4.13: The discrete plant in SystemC-AMS

4.3.2 The displacement input source

The test stimuli for testing the system consist of three different signals: constant signal, ramp signal and shaped signal. Here is the model of the input source in the following figure:

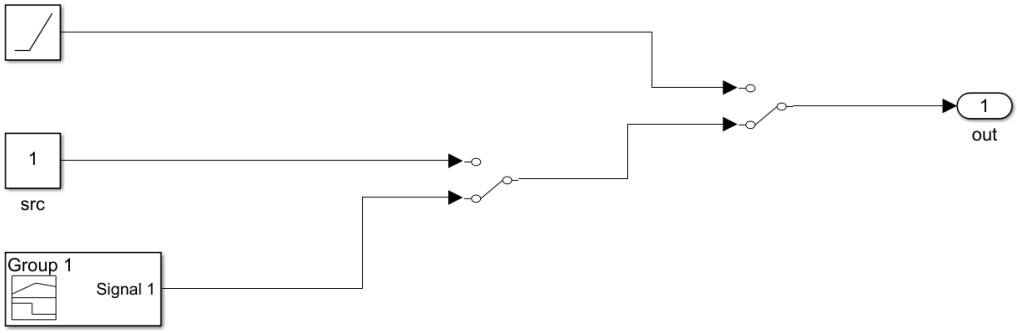

Figure 4.14: The displacement input in SystemC-AMS

The code for the displacement input source is in the following lines:

Listing 4.16: The displacement input source

```
//=================================================
//displacement_input_source.h
//=================================================
SCA_TDF_MODULE(displacement_input_source)
{
sca_tdf::sca_out<double> out;

displacement_input_source(sc_core::sc_module_name nm,
   sca_core::sca_time Tm = sca_core::sca_time(1, sc_core::
   SC_MS));

  void set_attributes();

  void processing();
private:
  double ampl; // amplitude
  double freq; // frequency
  sca_core::sca_time Tm; // module time step
};
//=================================================
```

Listing 4.17: The displacement input source

```
//=================================================
//displacement_input_source.cpp
//=================================================
//this is the displacement input source

#include "displacement_input_source.h"

displacement_input_source::displacement_input_source(sc_core
   ::sc_module_name nm, sca_core::sca_time Tm)

  : out("out"), ampl(1.0), freq(2), Tm(Tm)
{}

void displacement_input_source::set_attributes()
{
  set_timestep(Tm);
}

void displacement_input_source::processing()
{
  out.write(1.0);          //constant signal

//  double t = get_time().to_seconds(); // actual time
```

```
23 //       out.write(t);    //ramp signal
24
25 /*
26   double t = get_time().to_seconds(); // actual time
27
28   //shaped signal
29   if (t < 1)
30     out.write(0.0);
31 ...
32   else if (t >= 9 && t < 10)
33     out.write(-0.4 * t + 4);
34     */
35 }
36 //=====================================================
```

4.3.3 The discrete plant in the suspension system

The first suspension system consists of the discrete plant, constant damping coefficient and the displacement input source. The constant damping coefficient module is modeled using TDF. here is the model after connecting each module together:

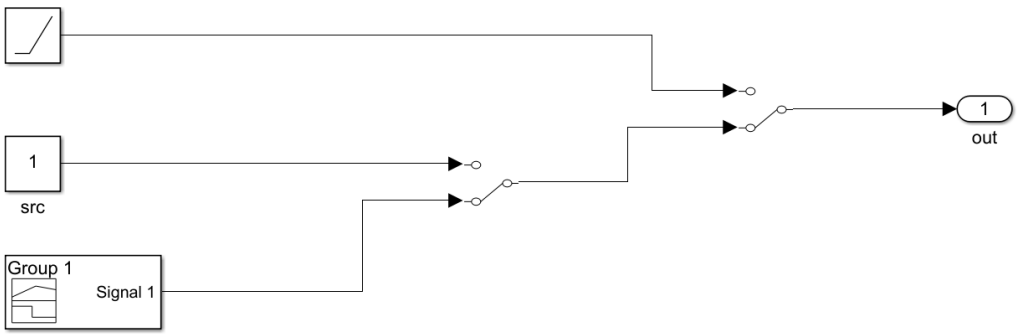

Figure 4.15: The displacement input for the suspension system in SystemC-AMS

4.3.4 Adding the controller to the harness

I modeled the skyhook controller using TDF. It has two inputs wheel and car velocities coming from the plant and provides the plant with the damping coefficient.

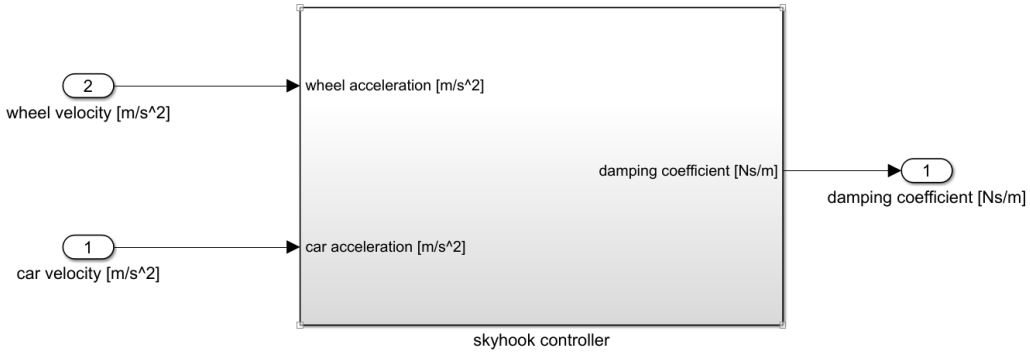

Figure 4.16: Skyhook controller, top layer in SystemC-AMS

Listing 4.18: The skyhook controller module

```
//================================================
//skyhook_controller.h
//================================================
SCA_TDF_MODULE(skyhook_controller)
{
sca_tdf::sca_in<double>  car_velocity;
  sca_tdf::sca_in<double>  wheel_velocity;
  sca_tdf::sca_out<double> damping_coefficient;

  skyhook_controller(sc_core::sc_module_name nm,
                     sca_core::sca_time Tm_ = sca_core::
   sca_time(1, sc_core::SC_MS) );

  void set_attributes();

  void processing();

private:
  double threshold; //(x1dot-x2dot) * x1dot
  double c1; //damping coefficient =6000
  double c2; //damping coefficient =1500
  sca_core::sca_time Tm; // module time step
};
//================================================
```

Listing 4.19: The skyhook controller module

```
//=====================================================
//skyhook_controller.cpp
//=============================================
skyhook_controller::skyhook_controller(sc_core::
    sc_module_name nm, sca_core::sca_time Tm)
  : car_velocity("car_velocity"), wheel_velocity("
    wheel_velocity"),
  damping_coefficient("damping_coefficient"), threshold(1),
    c1(6000), c2(1500), Tm(Tm)
{}

void skyhook_controller::set_attributes()
{
  set_timestep(Tm);
}

void skyhook_controller::processing()
{

  threshold = (car_velocity.read() - wheel_velocity.read())
    * car_velocity.read();
//  threshold = (wheel_velocity.read() - car_velocity.read()
    ) * wheel_velocity.read();
    if (threshold < 0.0)
      damping_coefficient.write(1500);
      else
      damping_coefficient.write(6000);
}
//=====================================================
```

4.3.5 Schedulability error

After connecting the controller directly to the system there is an error: System is not schedulable. It is due to cyclic dependencies. SystemC-AMS scheduler is responsible for managing the simulation of the different modules and their interactions within a system. It ensures that the simulation progresses in a synchronized and time-accurate manner. The error is due to that the scheduler assumes zero time delay between modules. This means that the output of one module is immediately available as an input to another module. However, if there are actual propagation delays between modules, they need to be explicitly modeled. I added a delay module between the controller and the plant to solve this error.

Listing 4.20: The delay module

```
//=================================================
```

```
2  //my_tdf_delay.h
3  //================================================
4  SCA_TDF_MODULE(my_tdf_delay) {
5    sca_tdf::sca_in<double> in;
6    sca_tdf::sca_out<double> out;
7    SCA_CTOR(my_tdf_delay) : in("in"), out("out") {}
8    void set_attributes()
9    {
10     set_timestep(1, sc_core::SC_MS);
11     out.set_delay(1);
12   }
13   void initialize()
14   {
15     out.initialize(1.1);
16   }
17   void processing()
18   {
19     out.write(in.read()); // directly write the input sample
       to the output (incl the delay)
20   }
21 };
22 //========================================================
```

After adding the delay model we have the complete system that contains the plant and the feedback controller as shown in the following figure:

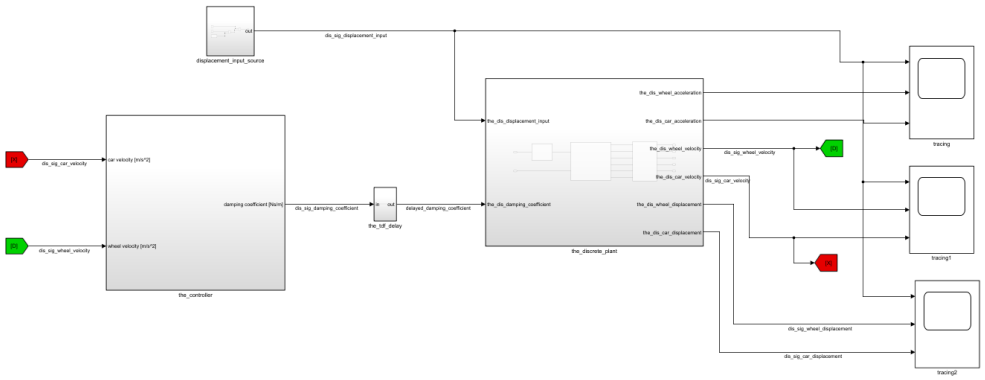

Figure 4.17: The suspension system after adding the delay model in SystemC-AMS

In order to test the plant in both conditions : with constant damping and by applying the controller I combined both modes in one model as shown in the following figure.

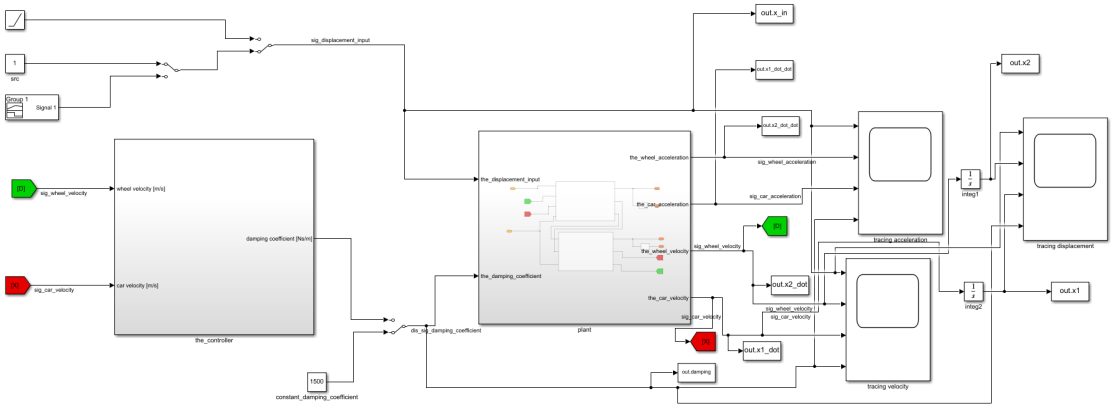

Figure 4.18: The final version of the suspension system in SystemC-AMS

In order to choose between the constant damping coefficient and the controller it could be done by connecting one of them directly to the plant or by adding another module called the damping switch. The implementation is shown in the following lines.

Listing 4.21: The damping switch

```
1 //=====================================================
2 void the_damping_switch::processing()
3 {
4   double my_switch = 0;
5   if (my_switch > 1)
6   damping_coefficient.write(first_damping.read());
7   else      damping_coefficient.write(first_damping.read());
8 }
9 //=====================================================
```

Now we have the complete system model, in next chapter we will see the simulation results and the comparison between SystemC-AMS and Simulink.

Chapter 5

Results

In this chapter I will show the results obtained from SystemC-AMS final model compared to Simulink results. The results of first three models of the plant and the whole system are identical. There are two types of results, the simulation results and the numerical ones. In the simulation result I applied three types of signal as a test stimuli for the displacement input: constant signal, ramp signal and shaped signal. We have two main type of results: first when damping coefficient is constant and the second when applying the controller. Each signal is tested with three different step size of the solver. For Simulink solver I have chosen Runge-Kutta fixed step solver. As we will see in the coming words, the displacement input obtained from SystemC-AMS is identical to Simulink in all cases, and for constant damping coefficient cases all results of damping coefficient are the same. For the numerical part I applied the same configurations for the simulations in order to calculate the error between SystemC-AMS and Simulink solvers.

5.1 Constant damping coefficient

In this part the controller is disabled, we have only the plant with constant damping coefficient to focus the analysis only on the mechanical aspects.

5.1.1 Constant displacement signal

I evaluated three different configurations by varying the time step from 10ms to 0.1ms. The first simulation runs at 10ms, and we can notice that the wheel acceleration of SystemC-AMS has the same starting point as in Simulink however, there is a small difference in first 0.25s, then they are almost identical. The same also for the car acceleration. This is caused by the sudden variation of the signals, behavior that is more sensitive to the adapted solver and to the characteristics of

the scheduling routine. Given the fixed time step configuration, both simulations estimate the variables under analysis in the same instants.

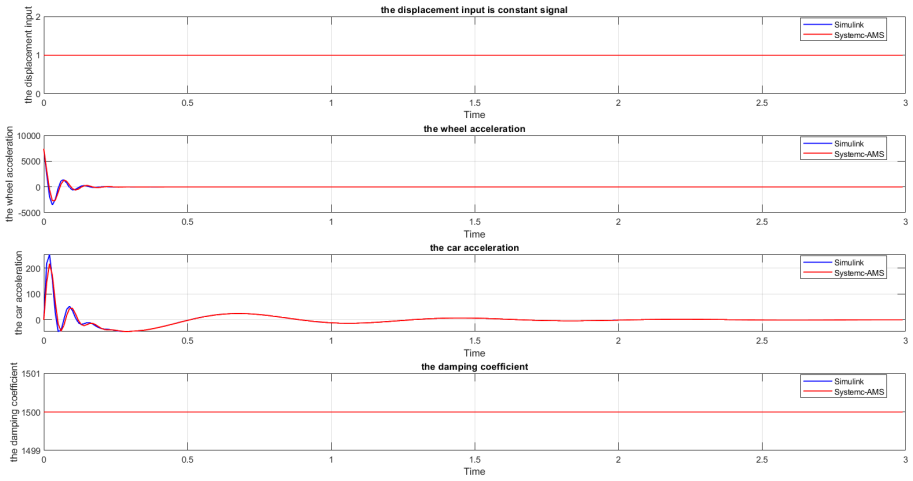

Figure 5.1: Constant signal, step size = 10ms, simulation time = 3s

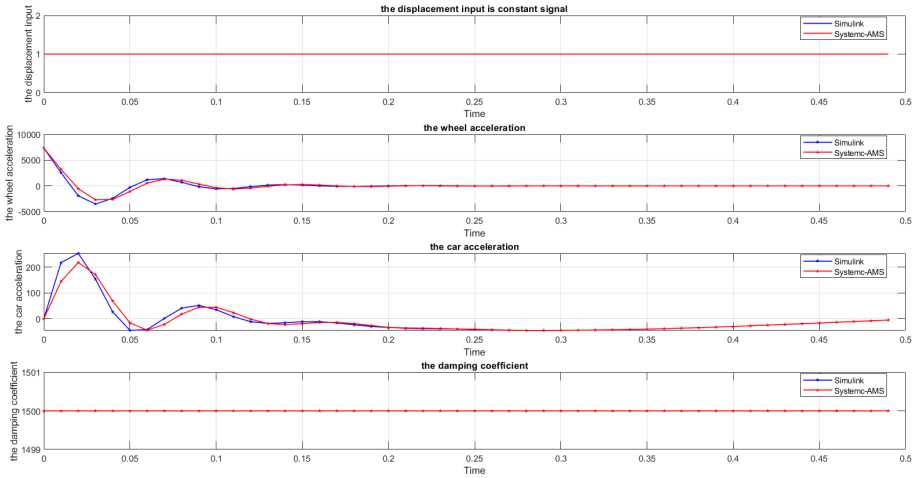

Figure 5.2: Constant signal, step size = 10ms, simulation time = 0.5s

Now we will see the error in both the wheel and car accelerations between SystemC-AMS and Simulink solvers in the first configuration when step size is 10ms. There are three types of errors: minimum, mean and max errors. For wheel

accelerations results, we can see that there are significantly large errors that are reduced while using higher order solvers. The error reaches to 100% in the case of Euler solver because it cannot simulate the model using the 10 ms step size. The same also for the car acceleration equation, however the error percent is better in the car acceleration results. This involves the sensitivity of the suspension system w.r.t. the adapted solver.

	ERROR (%) w.r.t. SystemC-AMS		
SIMULINK SOLVER	MIN	MEAN	MAX
ODE1b (backward Euler)	5.7265e-04%	12.6748%	23.2120%
ODE14x (extrapolation)	1.0940e-04%	7.4815%	19.2023%
ODE1 (Euler)	1.6036e-27%	100%	100%
ODE2 (Heun)	0.0020%	8.4118%	33.4597%
ODE3 (Bogacki-Shampine)	1.0771e-04%	6.6625%	16.1269%
ODE4 (Runge-Kutta)	1.0938e-04%	7.4847%	18.8359%
ODE5 (Dormand-Prince)	1.0941e-04%	7.4760%	19.1743%
ODE8 (Dormand-Prince)	1.0941e-04%	7.4809%	19.1976%

Table 5.1: Wheel acceleration results, constant signal, step size = 10ms, simulation time = 3s

It is important to note that the max error occurs only in the beginning of the simulation.

	ERROR (%) w.r.t. SystemC-AMS		
SIMULINK SOLVER	MIN	MEAN	MAX
ODE1b (backward Euler)	0	2.8287%	39.8036%
ODE14x (extrapolation)	0	1.4753%	29.0315%
ODE1 (Euler)	0	100%	5.6323e+03%
ODE2 (Heun)	0	0.1015%	42.8840%
ODE3 (Bogacki-Shampine)	0	1.3618%	27.3587%
ODE4 (Runge-Kutta)	0	1.5508%	28.7205%
ODE5 (Dormand-Prince)	0	1.4747%	29.0176%
ODE8 (Dormand-Prince)	0	1.4756%	29.0290%

Table 5.2: Car acceleration results, constant signal, step size = 10ms, simulation time = 3s

The second configuration reduces the step size to 1ms, the wheel and car

acceleration are almost identical. The error in wheel and car accelerations in this configuration is now reduced in all solvers, the mean error now is 0.0848% in case of Runge-Kutta solver for wheel acceleration equation.

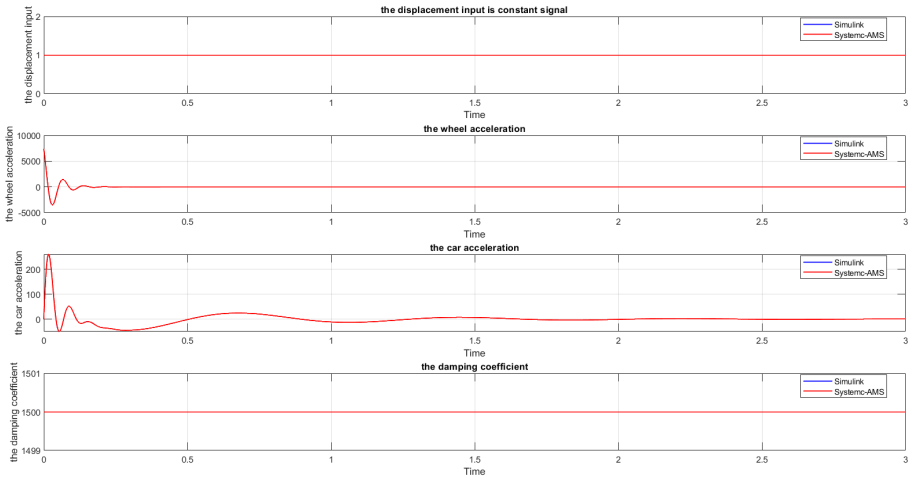

Figure 5.3: Constant signal, step size = 1ms, simulation time = 3s

	ERROR (%) w.r.t. SystemC-AMS		
SIMULINK SOLVER	MIN	MEAN	MAX
ODE1b (backward Euler)	0.0020%	2.0459%	5.3545%
ODE14x (extrapolation)	0.0014%	0.0847%	0.2408%
ODE1 (Euler)	0.0011%	1.6779%	6.5310%
ODE2 (Heun)	0.0011%	0.0577%	0.3133%
ODE3 (Bogacki-Shampine)	0.0014%	0.0838%	0.2373%
ODE4 (Runge-Kutta)	0.0014%	0.0848%	0.2408%
ODE5 (Dormand-Prince)	0.0014%	0.0847%	0.2408%
ODE8 (Dormand-Prince)	0.0014%	0.0847%	0.2408%

Table 5.3: Wheel acceleration results, constant signal, step size = 1ms, simulation time = 3s

	ERROR (%) w.r.t. SystemC-AMS		
SIMULINK SOLVER	MIN	MEAN	MAX
ODE1b (backward Euler)	0	0.0418%	7.0397%
ODE14x (extrapolation)	0	0.0352%	0.3835%
ODE1 (Euler)	0	0.0439%	7.9716%
ODE2 (Heun)	0	0.0202%	0.4122%
ODE3 (Bogacki-Shampine)	0	0.0353%	0.3809%
ODE4 (Runge-Kutta)	0	0.0352%	0.3835%
ODE5 (Dormand-Prince)	0	0.0352%	0.3835%
ODE8 (Dormand-Prince)	0	0.0352%	0.3835%

Table 5.4: Car acceleration results, constant signal, step size = 1ms, simulation time = 3s

In order to get more precise results, I reduced the fixed step size to be 0.1ms and the simulation results are identical. The errors between SystemC-AMS and Simulink decreased below 1% in most configurations.

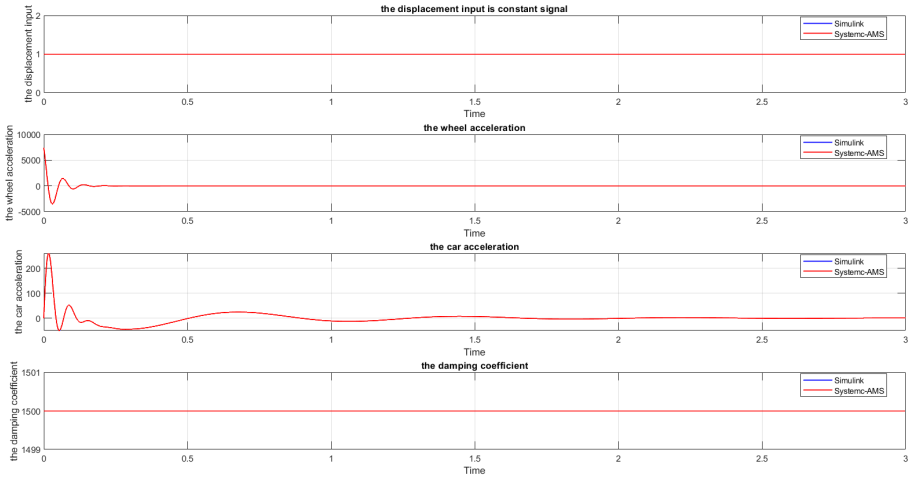

Figure 5.4: Constant signal, step size = 0.1ms, simulation time = 3s

	ERROR (%) w.r.t. SystemC-AMS		
SIMULINK SOLVER	MIN	MEAN	MAX
ODE1b (backward Euler)	0.0663%	0.1920%	0.5842%
ODE14x (extrapolation)	1.3061e-04%	7.7626e-04%	0.0028%
ODE1 (Euler)	0.0166%	0.1880%	0.5957%
ODE2 (Heun)	0.0012%	4.7283e-04%	0.0030%
ODE3 (Bogacki-Shampine)	2.2760e-04%	7.7534e-04%	0.0028%
ODE4 (Runge-Kutta)	1.6606e-04%	7.7627e-04%	0.0028%
ODE5 (Dormand-Prince)	1.7480e-04%	7.7626e-04%	0.0028%
ODE8 (Dormand-Prince)	1.7480e-04%	7.7626e-04%	0.0028%

Table 5.5: Wheel acceleration results, constant signal, step size = 0.1ms, simulation time = 3s

	ERROR (%) w.r.t. SystemC-AMS		
SIMULINK SOLVER	MIN	MEAN	MAX
ODE1b (backward Euler)	0	1.3799e-04%	0.7442%
ODE14x (extrapolation)	0	3.6776e-04%	0.0038%
ODE1 (Euler)	0	7.5285e-04%	0.7536%
ODE2 (Heun)	0	2.2048e-04%	0.0040%
ODE3 (Bogacki-Shampine)	0	3.6788e-04%	0.0038%
ODE4 (Runge-Kutta)	0	3.6776e-04%	0.0038%
ODE5 (Dormand-Prince)	0	3.6776e-04%	0.0038%
ODE8 (Dormand-Prince)	0	3.6776e-04%	0.0038%

Table 5.6: Car acceleration results, constant signal, step size = 0.1ms, simulation time = 3s

As a conclusion in the first case, it gives same results whenever the step size is very small. In first case when step size is 10ms, the Simulink results are better than SystemC-AMS results, as they are much more near to the results when the step size is smaller.

5.1.2 Ramp displacement signal

For the displacement input as ramp signal when step size is 10ms, same observation as in first case. There is a small difference in the first 0.25s, then they are almost identical in both wheel and car acceleration.

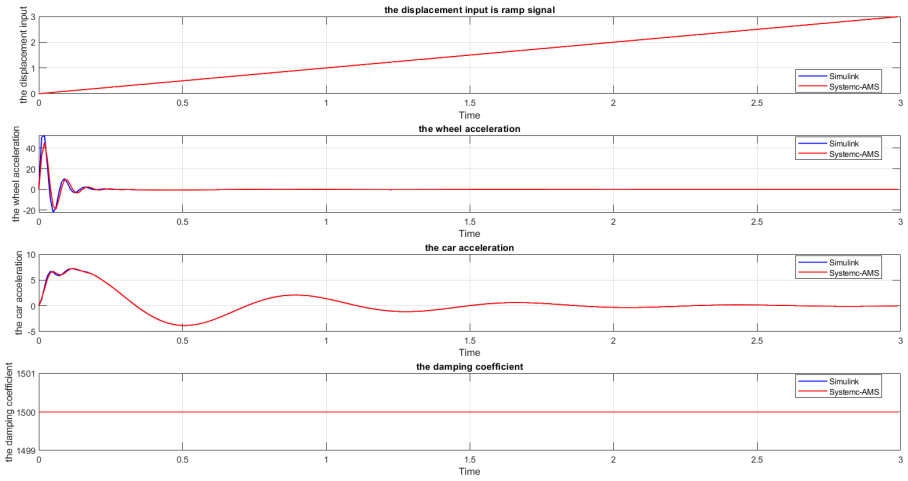

Figure 5.5: Ramp signal, step size = 10ms, simulation time = 3s

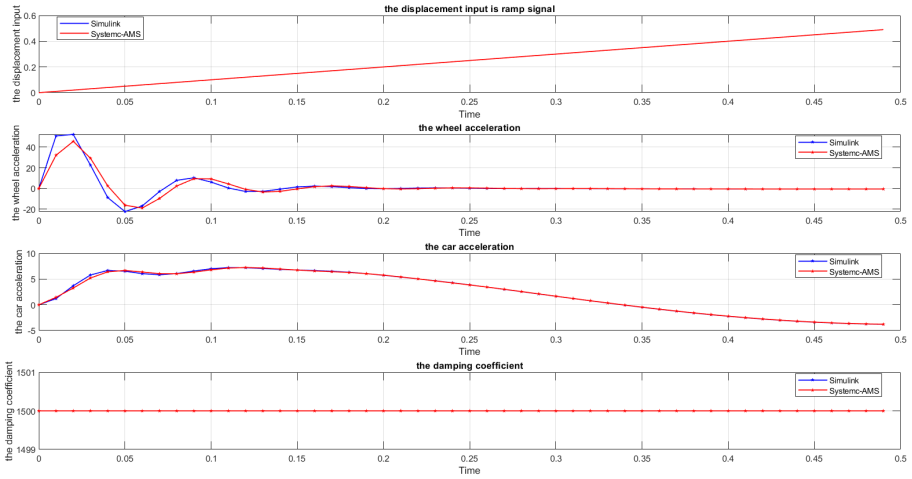

Figure 5.6: Ramp signal, step size = 10ms, simulation time = 0.5s

There are significant errors in both the wheel and car accelerations between SystemC-AMS and Simulink solvers in the second configuration when step size is 10 ms. These errors reduce in case of higher order solvers.

	ERROR (%) w.r.t. SystemC-AMS		
SIMULINK SOLVER	MIN	MEAN	MAX
ODE1b (backward Euler)	0	11.3388%	51.5278%
ODE14x (extrapolation)	0	3.8550%	35.9801%
ODE1 (Euler)	0	9.0958%	100%
ODE2 (Heun)	0	1.3688%	45.1598%
ODE3 (Bogacki-Shampine)	0	4.3404%	33.8096%
ODE4 (Runge-Kutta)	0	4.3744%	35.6213%
ODE5 (Dormand-Prince)	0	4.1638%	35.9646%
ODE8 (Dormand-Prince)	0	4.1623%	35.9794%

Table 5.7: Wheel acceleration results, ramp signal, step size = 10ms, simulation time = 3s

	ERROR (%) w.r.t. SystemC-AMS		
SIMULINK SOLVER	MIN	MEAN	MAX
ODE1b (backward Euler)	0	0.4349%	13.7949%
ODE14x (extrapolation)	0	0.1917%	8.3029%
ODE1 (Euler)	0	25.1934%	100%
ODE2 (Heun)	0	0.2957%	15.4096%
ODE3 (Bogacki-Shampine)	0	0.1630%	6.4536%
ODE4 (Runge-Kutta)	0	0.1963%	8.1445%
ODE5 (Dormand-Prince)	0	0.1985%	8.2871%
ODE8 (Dormand-Prince)	0	0.1988%	8.3012%

Table 5.8: Car acceleration results, ramp signal, step size = 10ms, simulation time = 3s

Applying the second configuration that reduces the step size to 1ms gives accurate results then, calculating the error, the mean error become 0.1180% and max error become 0.4725% in case of Runge-Kutta solver in case of wheel acceleration.

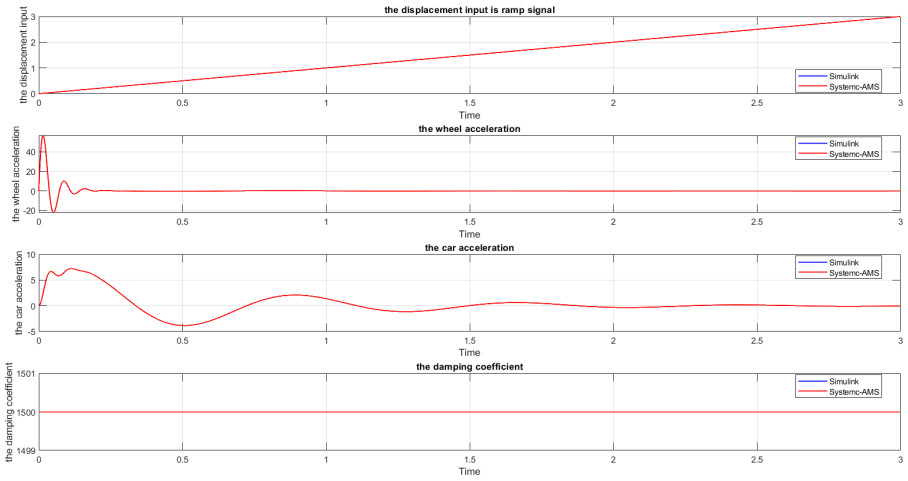

Figure 5.7: Ramp signal, step size = 1ms, simulation time = 3s

	ERROR (%) w.r.t. SystemC-AMS		
SIMULINK SOLVER	MIN	MEAN	MAX
ODE1b (backward Euler)	0	0.1668%	8.4152%
ODE14x (extrapolation)	0	0.1180%	0.4725%
ODE1 (Euler)	0	0.1198%	9.4093%
ODE2 (Heun)	0	0.0661%	0.4971%
ODE3 (Bogacki-Shampine)	0	0.1186%	0.4700%
ODE4 (Runge-Kutta)	0	0.1180%	0.4725%
ODE5 (Dormand-Prince)	0	0.1180%	0.4725%
ODE8 (Dormand-Prince)	0	0.1180%	0.4725%

Table 5.9: Wheel acceleration results, ramp signal, step size = 1ms, simulation time = 3s

	ERROR (%) w.r.t. SystemC-AMS		
SIMULINK SOLVER	MIN	MEAN	MAX
ODE1b (backward Euler)	0	0.0263%	0.0263%
ODE14x (extrapolation)	0	3.8232e-04%	0.1871%
ODE1 (Euler)	0	0.0272%	3.4055%
ODE2 (Heun)	0	4.9121e-04%	0.1849%
ODE3 (Bogacki-Shampine)	0	3.4676e-04%	0.1870%
ODE4 (Runge-Kutta)	0	3.8103e-04%	0.1871%
ODE5 (Dormand-Prince)	0	3.8084e-04%	0.1871%
ODE8 (Dormand-Prince)	0	3.8084e-04%	0.1871%

Table 5.10: Car acceleration results, ramp signal, step size = 1ms, simulation time = 3s

Then I reduced the step size to be 0.1 ms. The simulation results became more accurate and the errors between SystemC-AMS and Simulink decreased for all solvers.

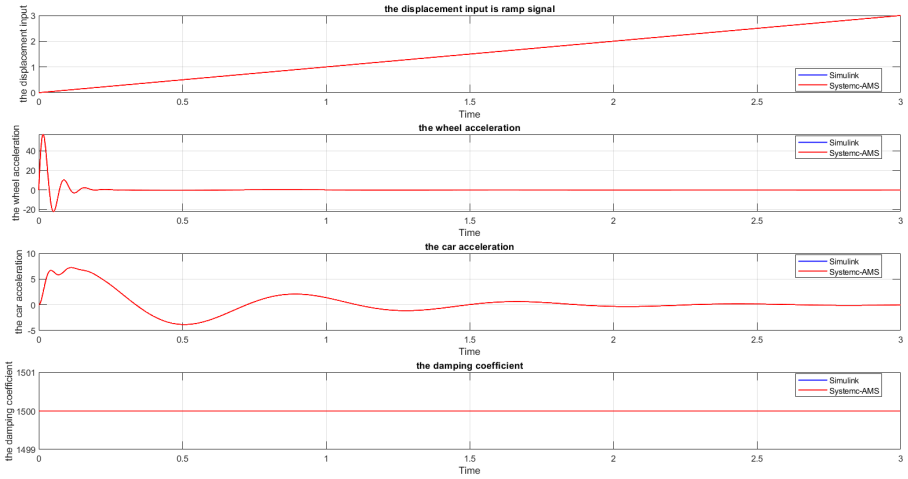

Figure 5.8: Ramp signal, step size = 0.1ms, simulation time = 3s

	ERROR (%) w.r.t. SystemC-AMS		
SIMULINK SOLVER	MIN	MEAN	MAX
ODE1b (backward Euler)	0	0.0019%	0.8858%
ODE14x (extrapolation)	0	0.0013%	0.0047%
ODE1 (Euler)	0	0.0011%	0.8957%
ODE2 (Heun)	0	7.4745e-04%	0.0049%
ODE3 (Bogacki-Shampine)	0	0.0013%	0.0047%
ODE4 (Runge-Kutta)	0	0.0013%	0.0047%
ODE5 (Dormand-Prince)	0	0.0013%	0.0047%
ODE8 (Dormand-Prince)	0	0.0013%	0.0047%

Table 5.11: Wheel acceleration results, ramp signal, step size = 0.1ms, simulation time = 3s

	ERROR (%) w.r.t. SystemC-AMS		
SIMULINK SOLVER	MIN	MEAN	MAX
ODE1b (backward Euler)	0	0.0027%	0.3121%
ODE14x (extrapolation)	0	3.5835e-07%	0.0020%
ODE1 (Euler)	0	0.0027%	0.3174%
ODE2 (Heun)	0	5.5152e-07%	0.0020%
ODE3 (Bogacki-Shampine)	0	3.2490e-07%	0.0020%
ODE4 (Runge-Kutta)	0	3.5826e-07%	0.0020%
ODE5 (Dormand-Prince)	0	3.5824e-07%	0.0020%
ODE8 (Dormand-Prince)	0	3.5824e-07%	0.0020%

Table 5.12: Car acceleration results, ramp signal, step size = 0.1ms, simulation time = 3s

5.1.3 Shaped displacement signal

Now, after applying the shaped signal with step size = 30ms, we can see that there is a difference between results of SystemC-AMS compared to Simulink. The car and wheel accelerations results of SystemC-AMS are better than Simulink results, since they are more accurate and more related to the results when step size is smaller.

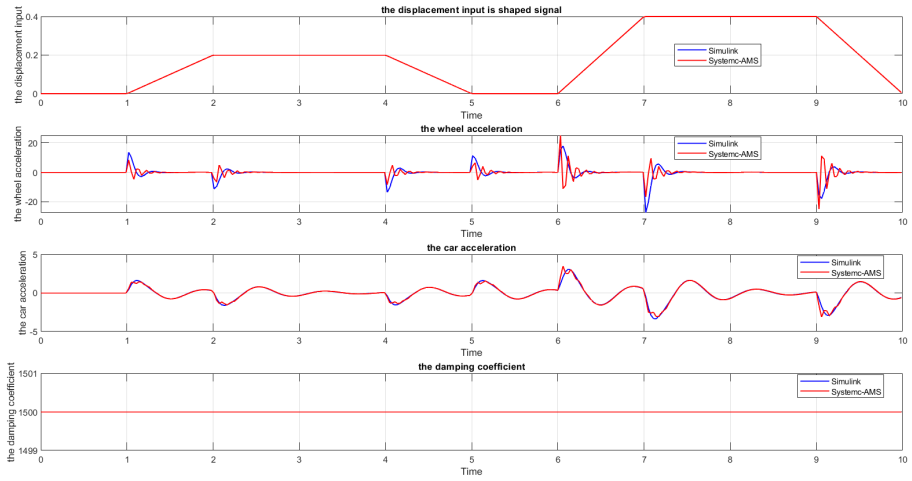

Figure 5.9: Shaped signal, step size = 30ms, simulation time = 10s

After reducing the step size to 10ms, the results are better however, there is a small difference each time when the displacement input signal is changed to another state, once again due to the different solvers and scheduling approach.

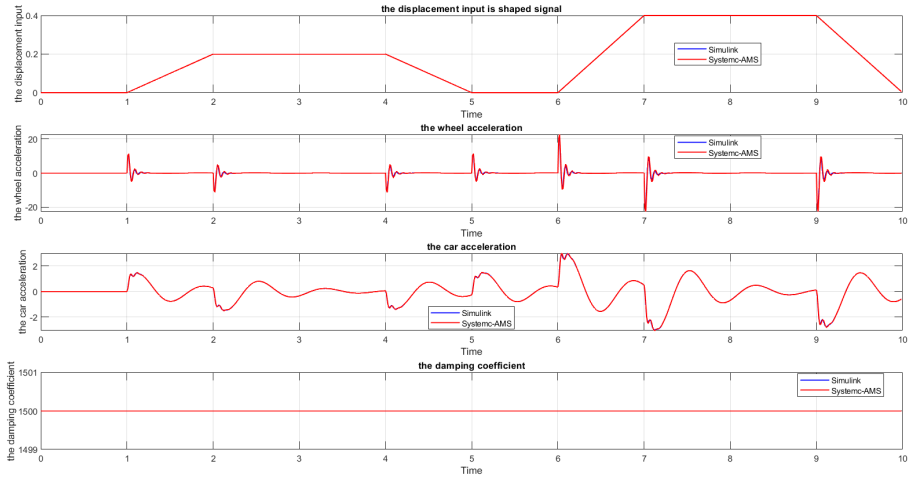

Figure 5.10: Shaped signal, step size = 10ms, simulation time = 10s

The mean wheel acceleration errors between SystemC-AMS and Simulink solvers do not exceed 1% in all solvers except of Euler solver while the max errors are large in all cases, even if restricted to the points of sudden displacement variation.

Same also for car acceleration errors.

	ERROR (%) w.r.t. SystemC-AMS		
SIMULINK SOLVER	MIN	MEAN	MAX
ODE1b (backward Euler)	0	0.1069%	74.7819%
ODE14x (extrapolation)	0	0.5318%	13.5552%
ODE1 (Euler)	0	25.7982%	100%
ODE2 (Heun)	0	1.7586%	42.6671%
ODE3 (Bogacki-Shampine)	0	0.6415%	18.1404%
ODE4 (Runge-Kutta)	0	0.4670%	12.0180%
ODE5 (Dormand-Prince)	0	0.5324%	13.5600%
ODE8 (Dormand-Prince)	0	0.5315%	13.5420%

Table 5.13: Wheel acceleration results, shaped signal, step size = 10ms, simulation time = 10s

	ERROR (%) w.r.t. SystemC-AMS		
SIMULINK SOLVER	MIN	MEAN	MAX
ODE1b (backward Euler)	0	0.3308%	21.7405%
ODE14x (extrapolation)	0	0.0026%	4.3281%
ODE1 (Euler)	0	9.2671%	100%
ODE2 (Heun)	0	0.0306%	13.5308%
ODE3 (Bogacki-Shampine)	0	0.0058%	5.3287%
ODE4 (Runge-Kutta)	0	0.0020%	3.8929%
ODE5 (Dormand-Prince)	0	0.0025%	4.3346%
ODE8 (Dormand-Prince)	0	0.0026%	4.3258%

Table 5.14: Car acceleration results, shaped signal, step size = 10ms, simulation time = 10s

Reducing the step size to be 1ms increased the accuracy of the simulation results as shown in the following figure:

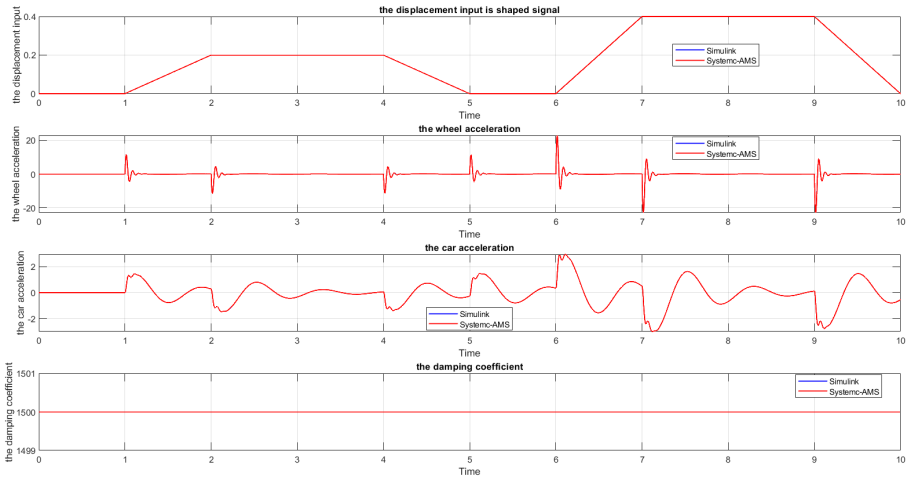

Figure 5.11: Shaped signal, step size = 1ms, simulation time = 10s

Mean and max errors between SystemC-AMS and Simulink solvers are reduced, for car acceleration mean error in case of Runge-Kutta solver it become 0.0051% and max error become 0.1267% while for car acceleration max error become 0.0439% .

	ERROR (%) w.r.t. SystemC-AMS		
SIMULINK SOLVER	MIN	MEAN	MAX
ODE1b (backward Euler)	0	0.0066%	8.6075%
ODE14x (extrapolation)	0	0.0051%	0.1268%
ODE1 (Euler)	0	0.0055%	9.2339%
ODE2 (Heun)	0	0.0158%	0.3869%
ODE3 (Bogacki-Shampine)	0	0.0050%	0.1250%
ODE4 (Runge-Kutta)	0	0.0051%	0.1267%
ODE5 (Dormand-Prince)	0	0.0051%	0.1268%
ODE8 (Dormand-Prince)	0	0.0051%	0.1268%

Table 5.15: Wheel acceleration results, shaped signal, step size = 1ms, simulation time = 10s

	ERROR (%) w.r.t. SystemC-AMS		
SIMULINK SOLVER	MIN	MEAN	MAX
ODE1b (backward Euler)	0	0.0337%	3.1061%
ODE14x (extrapolation)	0	2.4932e-05%	0.0440%
ODE1 (Euler)	0	0.0334%	3.4516%
ODE2 (Heun)	0	9.8674e-05%	0.1331%
ODE3 (Bogacki-Shampine)	0	1.6901e-05%	0.0437%
ODE4 (Runge-Kutta)	0	2.4972e-05%	0.0439%
ODE5 (Dormand-Prince)	0	2.4930e-05%	0.0440%
ODE8 (Dormand-Prince)	0	2.4930e-05%	0.0440%

Table 5.16: Car acceleration results, shaped signal, step size = 1ms, simulation time = 10s

Then after reducing the step size to be 0.1ms the simulation results become more accurate, with errors lower than 0.1 for most configurations.

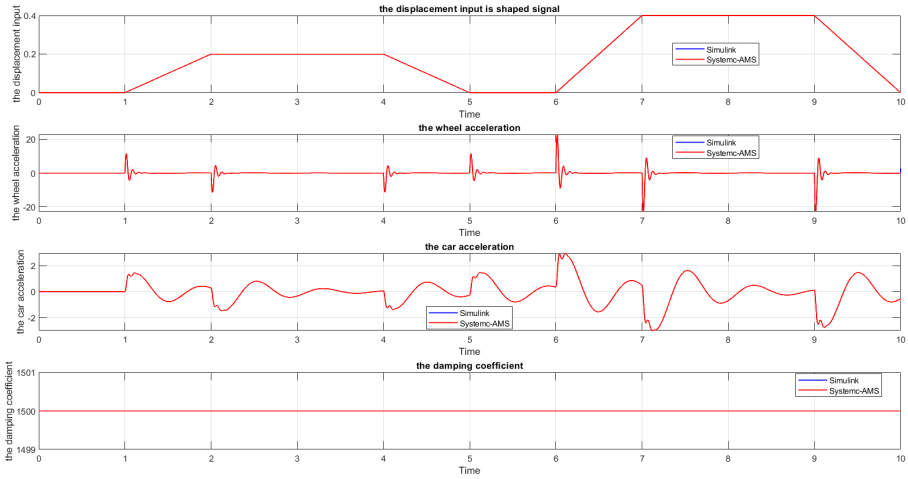

Figure 5.12: Shaped signal, step size = 0.1ms, simulation time = 10s

	ERROR (%) w.r.t. SystemC-AMS		
SIMULINK SOLVER	MIN	MEAN	MAX
ODE1b (backward Euler)	0	0.0282%	11.3875%
ODE14x (extrapolation)	0	0.0274%	11.3502%
ODE1 (Euler)	0	0.0265%	11.3122%
ODE2 (Heun)	0	0.0275%	11.3504%
ODE3 (Bogacki-Shampine)	0	0.0274%	11.3502%
ODE4 (Runge-Kutta)	0	0.0274%	11.3502%
ODE5 (Dormand-Prince)	0	0.0274%	11.3502%
ODE8 (Dormand-Prince)	0	0.0274%	11.3502%

Table 5.17: Wheel acceleration results, shaped signal, step size = 0.1ms, simulation time = 10s

	ERROR (%) w.r.t. SystemC-AMS		
SIMULINK SOLVER	MIN	MEAN	MAX
ODE1b (backward Euler)	0	0.0034%	0.3262%
ODE14x (extrapolation)	0	2.5565e-05%	0.1553%
ODE1 (Euler)	0	0.0033%	0.3297%
ODE2 (Heun)	0	2.5051e-05%	0.1554%
ODE3 (Bogacki-Shampine)	0	2.5572e-05%	0.1553%
ODE4 (Runge-Kutta)	0	2.5565e-05%	0.1553%
ODE5 (Dormand-Prince)	0	2.5565e-05%	0.1553%
ODE8 (Dormand-Prince)	0	2.5565e-05%	0.1553%

Table 5.18: Car acceleration results, shaped signal, step size = 0.1ms, simulation time = 10s

5.2 Variable damping coefficient

In this part the controller is enabled, we have now the complete suspension system: the plant plus the Skyhook controller that changes the damping coefficient according to wheel and car acceleration.

5.2.1 Constant displacement signal

As in the first part I evaluated three different configurations by varying the time step from 10ms to 0.1ms. The first simulation runs at 10ms, there is a difference between the car and wheel acceleration in first 0.5s, then they are almost identical in both SystemC-AMS and Simulink. For the damping coefficient, the time response to change the damping coefficient value is different in both of SystemC-AMS and Simulink.

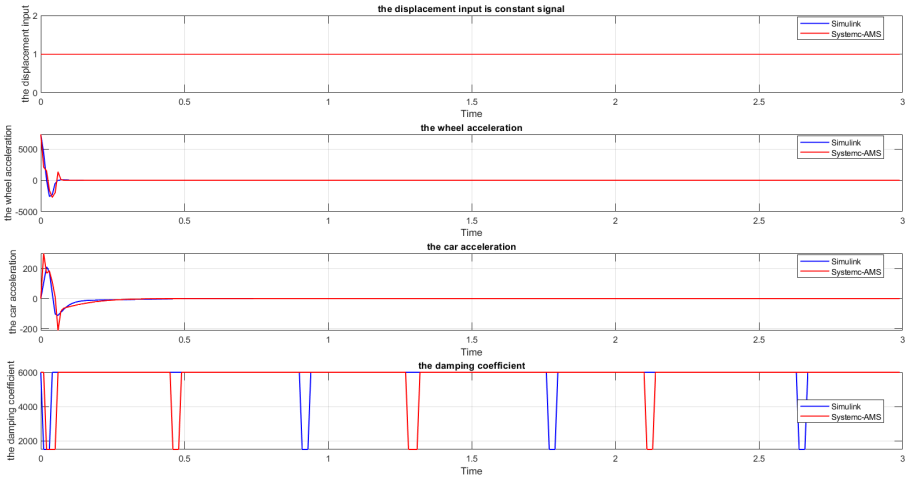

Figure 5.13: Constant signal, step size = 10ms, simulation time = 3s

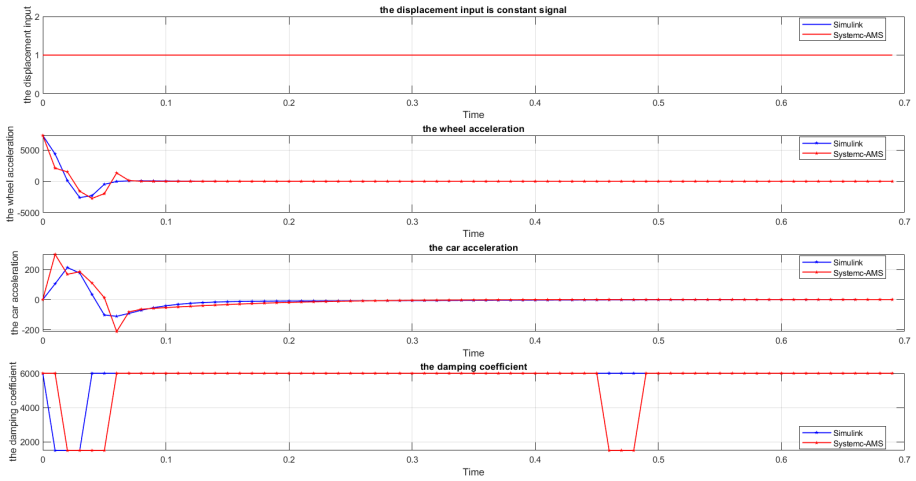

Figure 5.14: Constant signal, step size = 10ms, simulation time = 0.7s

Now we will see the error in the wheel, car acceleration and damping coefficient between SystemC-AMS and Simulink solvers in the first configuration when step size is 10 ms. For wheel accelerations results we can see that there are significant large errors that are reduced while using higher order solvers. The mean error reaches to 23.4211% in the case of Euler solver and Backward Euler because it cannot simulate the model using the 10 ms step size. The error of car acceleration is also high in most of solvers. Damping coefficient mean error is near to 1% , and damping coefficient max error reaches 75%, however it is accepted since it means that there is a difference in the time response between SystemC-AMS and Simulink. This is caused by the high sensitivity of the controller w.r.t. the other estimated variables. Even a small error may impact on it cause determining the value of the damping coefficient.

	ERROR (%) w.r.t. SystemC-AMS		
SIMULINK SOLVER	MIN	MEAN	MAX
ODE1b (backward Euler)	0	23.4502%	34.9694%
ODE14x (extrapolation)	0	12.4372%	47.1583%
ODE1 (Euler)	0	23.4211%	122.3141%
ODE2 (Heun)	0	2.8393%	38.8193%
ODE3 (Bogacki-Shampine)	0	2.5621%	31.5235%
ODE4 (Runge-Kutta)	0	3.4936%	30.8468%
ODE5 (Dormand-Prince)	0	4.4226%	25.8617%
ODE8 (Dormand-Prince)	0	12.9851%	47.1909%

Table 5.19: Wheel acceleration results, constant signal, step size = 10ms, simulation time = 3s

	ERROR (%) w.r.t. SystemC-AMS		
SIMULINK SOLVER	MIN	MEAN	MAX
ODE1b (backward Euler)	0	7.5728%	136.7049%
ODE14x (extrapolation)	0	8.3601%	77.1123%
ODE1 (Euler)	0	5.1702%	74.1266%
ODE2 (Heun)	0	21.6351%	200.1473%
ODE3 (Bogacki-Shampine)	0	2.2003%	79.3600%
ODE4 (Runge-Kutta)	0	9.3666%	91.9307%
ODE5 (Dormand-Prince)	0	10.4230%	84.3219%
ODE8 (Dormand-Prince)	0	8.9170%	74.3118%

Table 5.20: Car acceleration results, constant signal, step size = 10ms, simulation time = 3s

	ERROR (%) w.r.t. SystemC-AMS		
SIMULINK SOLVER	MIN	MEAN	MAX
ODE1b (backward Euler)	0	0.5222%	75%
ODE14x (extrapolation)	0	1.0283%	75%
ODE1 (Euler)	0	0.2591%	75%
ODE2 (Heun)	0	1.7857%	75%
ODE3 (Bogacki-Shampine)	0	1.0283%	75%
ODE4 (Runge-Kutta)	0	0.7732%	75%
ODE5 (Dormand-Prince)	0	1.0283%	75%
ODE8 (Dormand-Prince)	0	0.7732%	75%

Table 5.21: Damping coefficient results, constant signal, step size = 10ms, simulation time = 3s

Then applying the second configuration, reducing step size to 1ms, the wheel and car accelerations are almost identical. The error in wheel and car accelerations in this configuration is reduced in all solvers, the mean error is 0.2721% in case of Runge-Kutta solver for wheel acceleration, in case of car acceleration the mean error is 0.2070% and the mean error in the damping coefficient becomes 5.3241% .

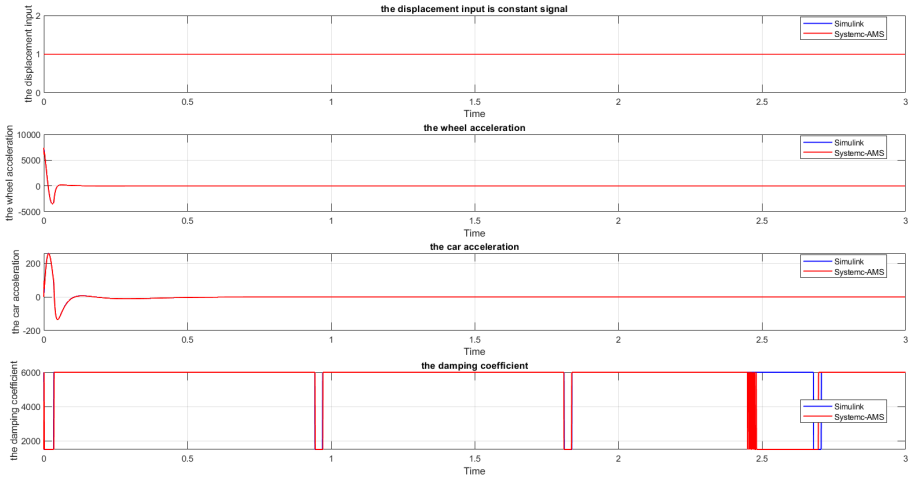

Figure 5.15: Constant signal, step size = 1ms, simulation time = 3s

	ERROR (%) w.r.t. SystemC-AMS		
SIMULINK SOLVER	MIN	MEAN	MAX
ODE1b (backward Euler)	0	3.2275%	11.8797%
ODE14x (extrapolation)	0	0.4149%	11.4681%
ODE1 (Euler)	0	2.9989%	6.4412%
ODE2 (Heun)	0	0.0875%	11.8366%
ODE3 (Bogacki-Shampine)	0	0.2766%	11.6082%
ODE4 (Runge-Kutta)	0	0.2721%	11.6011%
ODE5 (Dormand-Prince)	0	0.1383%	11.7392%
ODE8 (Dormand-Prince)	0	0.4161%	11.4681%

Table 5.22: Wheel acceleration results, constant signal, step size = 1ms, simulation time = 3s

	ERROR (%) w.r.t. SystemC-AMS		
SIMULINK SOLVER	MIN	MEAN	MAX
ODE1b (backward Euler)	0	0.3676%	28.9258%
ODE14x (extrapolation)	0	0.3223%	25.8454%
ODE1 (Euler)	0	0.3167%	17.3786%
ODE2 (Heun)	0	0.0531%	26.7533%
ODE3 (Bogacki-Shampine)	0	0.2124%	26.1318%
ODE4 (Runge-Kutta)	0	0.2070%	26.1373%
ODE5 (Dormand-Prince)	0	0.0988%	26.4668%
ODE8 (Dormand-Prince)	0	0.3233%	25.8454%

Table 5.23: Car acceleration results, constant signal, step size = 1ms, simulation time = 3s

	ERROR (%) w.r.t. SystemC-AMS		
SIMULINK SOLVER	MIN	MEAN	MAX
ODE1b (backward Euler)	0	4.8345%	75%
ODE14x (extrapolation)	0	0.7665%	75%
ODE1 (Euler)	0	5.2753%	75%
ODE2 (Heun)	0	5.3241%	75%
ODE3 (Bogacki-Shampine)	0	4.5000%	75%
ODE4 (Runge-Kutta)	0	5.3241%	75%
ODE5 (Dormand-Prince)	0	5.2510%	75%
ODE8 (Dormand-Prince)	0	5.2997%	75%

Table 5.24: Damping coefficient results, constant signal, step size = 1ms, simulation time = 3s

Then applying the third configuration that reduce the step size to be 0.1ms, the simulation results are almost identical. The errors between SystemC-AMS and Simulink decreased to a single digit.

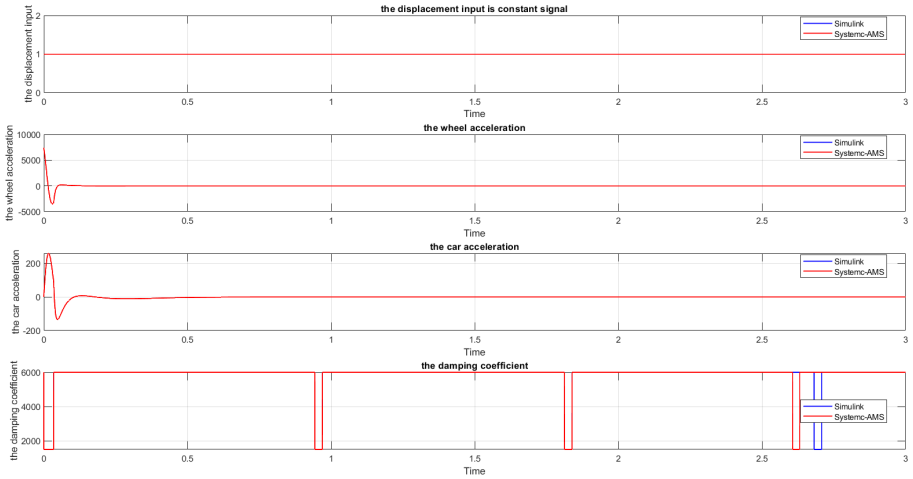

Figure 5.16: Constant signal, step size = 0.1ms, simulation time = 3s

	ERROR (%) w.r.t. SystemC-AMS		
SIMULINK SOLVER	MIN	MEAN	MAX
ODE1b (backward Euler)	0	0.2835%	1.4212%
ODE14x (extrapolation)	0	0.0053%	1.4170%
ODE1 (Euler)	0	0.2828%	0.5960%
ODE2 (Heun)	0	0.0015%	1.4208%
ODE3 (Bogacki-Shampine)	0	0.0039%	1.4183%
ODE4 (Runge-Kutta)	0	0.0039%	1.4183%
ODE5 (Dormand-Prince)	0	0.0023%	1.4198%
ODE8 (Dormand-Prince)	0	0.0052%	1.4170%

Table 5.25: Wheel acceleration results, constant signal, step size = 0.1ms, simulation time = 3s

	ERROR (%) w.r.t. SystemC-AMS		
SIMULINK SOLVER	MIN	MEAN	MAX
ODE1b (backward Euler)	0	0.0027%	3.2926%
ODE14x (extrapolation)	0	0.0042%	3.2566%
ODE1 (Euler)	0	0.0041%	2.4679%
ODE2 (Heun)	0	0.0012%	3.2654%
ODE3 (Bogacki-Shampine)	0	0.0031%	3.2595%
ODE4 (Runge-Kutta)	0	0.0031%	3.2595%
ODE5 (Dormand-Prince)	0	0.0018%	3.2630%
ODE8 (Dormand-Prince)	0	0.0041%	3.2566%

Table 5.26: Car acceleration results, constant signal, step size = 0.1ms, simulation time = 3s

	ERROR (%) w.r.t. SystemC-AMS		
SIMULINK SOLVER	MIN	MEAN	MAX
ODE1b (backward Euler)	0	0.0283%	75%
ODE14x (extrapolation)	0	0.6569%	75%
ODE1 (Euler)	0	0.0257%	75%
ODE2 (Heun)	0	0.0283%	75%
ODE3 (Bogacki-Shampine)	0	0.0283%	75%
ODE4 (Runge-Kutta)	0	0.0283%	75%
ODE5 (Dormand-Prince)	0	0.0283%	75%
ODE8 (Dormand-Prince)	0	0.0257%	75%

Table 5.27: Damping coefficient results, constant signal, step size = 0.1ms, simulation time = 3s

5.2.2 Ramp displacement signal

For the displacement input as ramp signal when step size is 10 ms, there is a small difference in first 0.25s then, they are almost identical in both wheel and car acceleration.

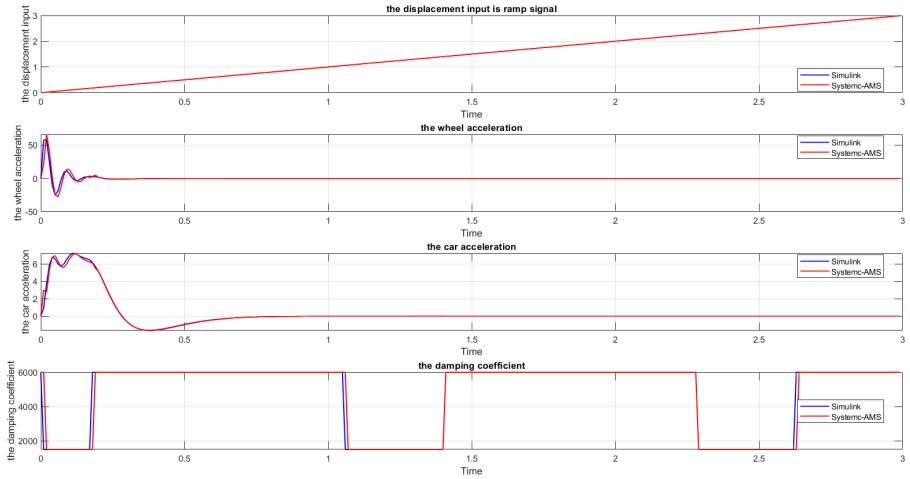

Figure 5.17: Ramp signal, step size = 10ms, simulation time = 3s

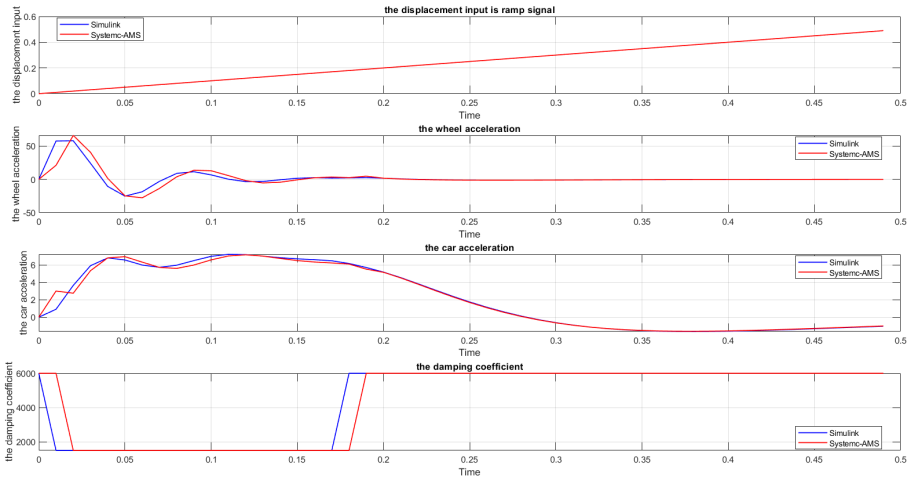

Figure 5.18: Ramp signal, step size = 10ms, simulation time = 0.5s

There are a significant errors in both the wheel and car accelerations between SystemC-AMS and Simulink solvers in the second configuration when step size is 10ms. The damping coefficient error is accepted in this case.

	ERROR (%) w.r.t. SystemC-AMS		
SIMULINK SOLVER	MIN	MEAN	MAX
ODE1b (backward Euler)	0	21.8172%	55.4656%
ODE14x (extrapolation)	0	2.0482%	57.2456%
ODE1 (Euler)	0	0.8735%	98.3391%
ODE2 (Heun)	0	21.5311%	39.5893%
ODE3 (Bogacki-Shampine)	0	10.4101%	59.7814%
ODE4 (Runge-Kutta)	0	5.3897%	62.9323%
ODE5 (Dormand-Prince)	0	6.3136%	61.9752%
ODE8 (Dormand-Prince)	0	6.7062%	60.0913%

Table 5.28: Wheel acceleration results, ramp signal, step size = 10ms, simulation time = 3s

	ERROR (%) w.r.t. SystemC-AMS		
SIMULINK SOLVER	MIN	MEAN	MAX
ODE1b (backward Euler)	0	0.4772%	32.5521%
ODE14x (extrapolation)	0	0.8185%	24.6204%
ODE1 (Euler)	0	0.6201%	121.4343%
ODE2 (Heun)	0	3.4782%	38.9378%
ODE3 (Bogacki-Shampine)	0	0.3391%	34.8066%
ODE4 (Runge-Kutta)	0	0.4963%	28.9746%
ODE5 (Dormand-Prince)	0	0.2988%	30.1850%
ODE8 (Dormand-Prince)	0	0.2213%	30.8886%

Table 5.29: Car acceleration results, ramp signal, step size = 10ms, simulation time = 3s

	ERROR (%) w.r.t. SystemC-AMS		
SIMULINK SOLVER	MIN	MEAN	MAX
ODE1b (backward Euler)	0	0.8128%	75%
ODE1b (backward Euler)	0	5.0336%	75%
ODE14x (extrapolation)	0	8.7464%	75%
ODE1 (Euler)	0	5.4381%	75%
ODE2 (Heun)	0	0.6349%	75%
ODE3 (Bogacki-Shampine)	0	0.3185%	75%
ODE4 (Runge-Kutta)	0	0.3185%	75%
ODE5 (Dormand-Prince)	0	0.3185%	75%
ODE8 (Dormand-Prince)	0	0.3185%	75%

Table 5.30: Damping coefficient results, ramp signal, step size = 10ms, simulation time = 3s

Then we have the second configuration that reduces the step size to 1ms, it gives accurate results then, calculating the error, the mean error of wheel acceleration become 0.0.1062% and max error become 1.8257% in case of Runge-Kutta solver.

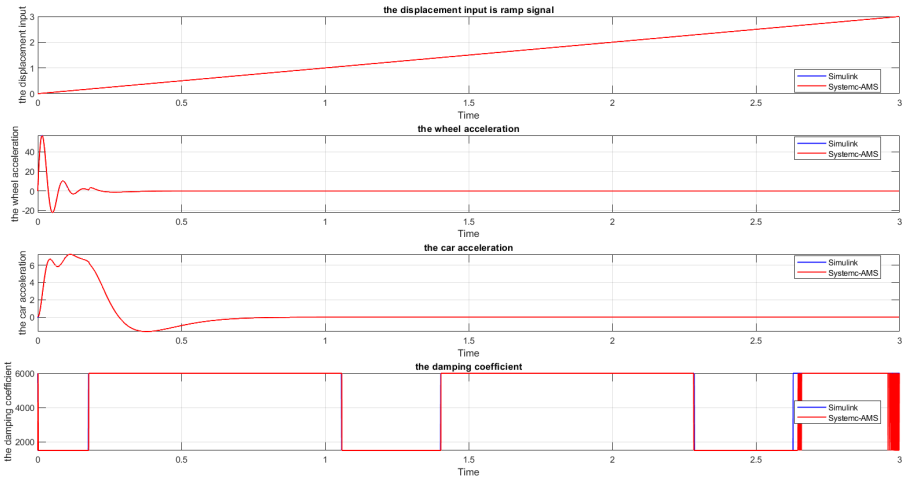

Figure 5.19: Ramp signal, step size = 1ms, simulation time = 3s

	ERROR (%) w.r.t. SystemC-AMS		
SIMULINK SOLVER	MIN	MEAN	MAX
ODE1b (backward Euler)	0	0.1946%	8.4386%
ODE14x (extrapolation)	0	0.1551%	1.8173%
ODE1 (Euler)	0	0.1125%	9.3977%
ODE2 (Heun)	0	0.0294%	0.8924%
ODE3 (Bogacki-Shampine)	0	0.1122%	1.8329%
ODE4 (Runge-Kutta)	0	0.1062%	1.8257%
ODE5 (Dormand-Prince)	0	0.1046%	1.8194%
ODE8 (Dormand-Prince)	0	0.1062%	1.8243%

Table 5.31: Wheel acceleration results, ramp signal, step size = 1ms, simulation time = 3s

	ERROR (%) w.r.t. SystemC-AMS		
SIMULINK SOLVER	MIN	MEAN	MAX
ODE1b (backward Euler)	0	0.0018%	2.8994%
ODE14x (extrapolation)	0	0.0030%	1.1252%
ODE1 (Euler)	0	0.0027%	3.4061%
ODE2 (Heun)	0	0.0052%	0.5216%
ODE3 (Bogacki-Shampine)	0	0.0017%	1.1356%
ODE4 (Runge-Kutta)	0	0.0024%	1.1303%
ODE5 (Dormand-Prince)	0	0.0025%	1.1264%
ODE8 (Dormand-Prince)	0	0.0024%	1.1297%

Table 5.32: Car acceleration results, ramp signal, step size = 1ms, simulation time = 3s

	ERROR (%) w.r.t. SystemC-AMS		
SIMULINK SOLVER	MIN	MEAN	MAX
ODE1b (backward Euler)	0	4.4932%	75%
ODE14x (extrapolation)	0	3.5487%	75%
ODE1 (Euler)	0	1.4340%	75%
ODE2 (Heun)	0	1.4026%	75%
ODE3 (Bogacki-Shampine)	0	1.3397%	75%
ODE4 (Runge-Kutta)	0	1.3397%	75%
ODE5 (Dormand-Prince)	0	1.3397%	75%
ODE8 (Dormand-Prince)	0	1.3397%	75%

Table 5.33: Damping coefficient results, ramp signal, step size = 1ms, simulation time = 3s

Applying the third configuration that reduce the step size to be 0.1ms. The simulation results became almost in the same level of accuracy and the errors between SystemC-AMS and Simulink decreased for all solvers.

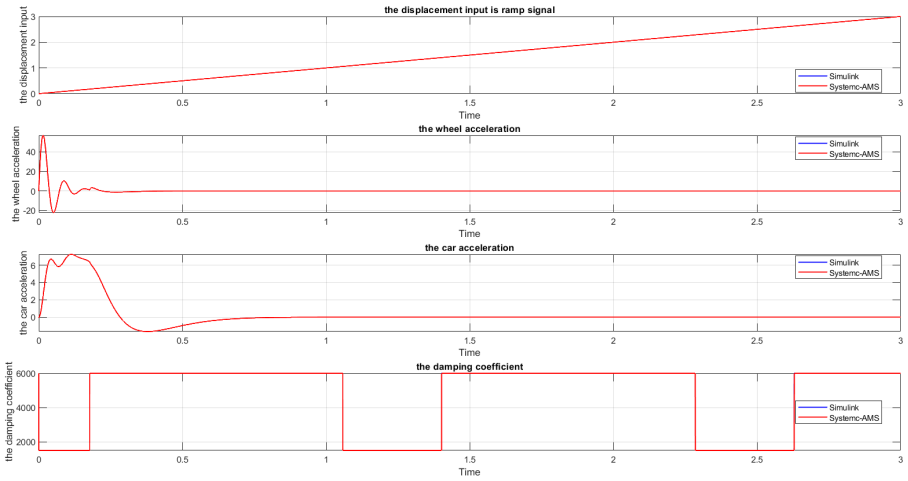

Figure 5.20: Ramp signal, step size = 0.1ms, simulation time = 3s

	ERROR (%) w.r.t. SystemC-AMS		
SIMULINK SOLVER	MIN	MEAN	MAX
ODE1b (backward Euler)	0	0.0017%	0.8858%
ODE14x (extrapolation)	0	0.0011%	0.0215%
ODE1 (Euler)	0	0.0015%	0.8957%
ODE2 (Heun)	0	6.4592e-04%	0.0121%
ODE3 (Bogacki-Shampine)	0	0.0012%	0.0215%
ODE4 (Runge-Kutta)	0	0.0012%	0.0215%
ODE5 (Dormand-Prince)	0	0.0012%	0.0215%
ODE8 (Dormand-Prince)	0	0.0012%	0.0215%

Table 5.34: Wheel acceleration results, ramp signal, step size = 0.1ms, simulation time = 3s

	ERROR (%) w.r.t. SystemC-AMS		
SIMULINK SOLVER	MIN	MEAN	MAX
ODE1b (backward Euler)	0	1.4908e-05%	0.3121%
ODE14x (extrapolation)	0	1.7061e-05%	0.0137%
ODE1 (Euler)	0	3.2102e-06%	0.3174%
ODE2 (Heun)	0	1.3742e-05%	0.0077%
ODE3 (Bogacki-Shampine)	0	4.7176e-06%	0.0137%
ODE4 (Runge-Kutta)	0	5.4227e-06%	0.0137%
ODE5 (Dormand-Prince)	0	5.6911e-06%	0.0137%
ODE8 (Dormand-Prince)	0	5.3044e-06%	0.0137%

Table 5.35: Car acceleration results, ramp signal, step size = 0.1ms, simulation time = 3s

	ERROR (%) w.r.t. SystemC-AMS		
SIMULINK SOLVER	MIN	MEAN	MAX
ODE1b (backward Euler)	0	2.7874%	75%
ODE14x (extrapolation)	0	13.2370%	75%
ODE1 (Euler)	0	0.0191%	75%
ODE2 (Heun)	0	0.0191%	75%
ODE3 (Bogacki-Shampine)	0	0.0159%	75%
ODE4 (Runge-Kutta)	0	0.0159%	75%
ODE5 (Dormand-Prince)	0	0.0159%	75%
ODE8 (Dormand-Prince)	0	0.0159%	75%

Table 5.36: Damping coefficient results, ramp signal, step size = 0.1ms, simulation time = 3s

5.2.3 Shaped displacement signal

For shaped displacement signal with step size = 30ms, with variable damping coefficient, Simulink was unable to generate the simulation with this large step size. SystemC-AMS was able to generate the simulation with this configuration.

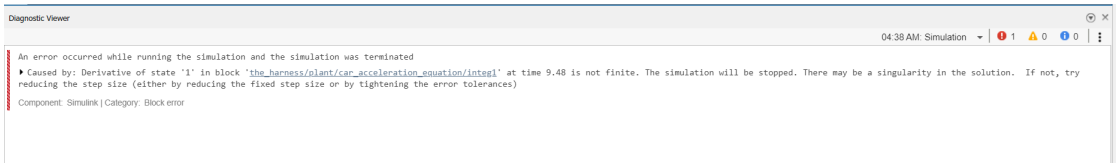

Figure 5.21: Error in Simulink when shaped signal, step size = 30ms, simulation time = 10s

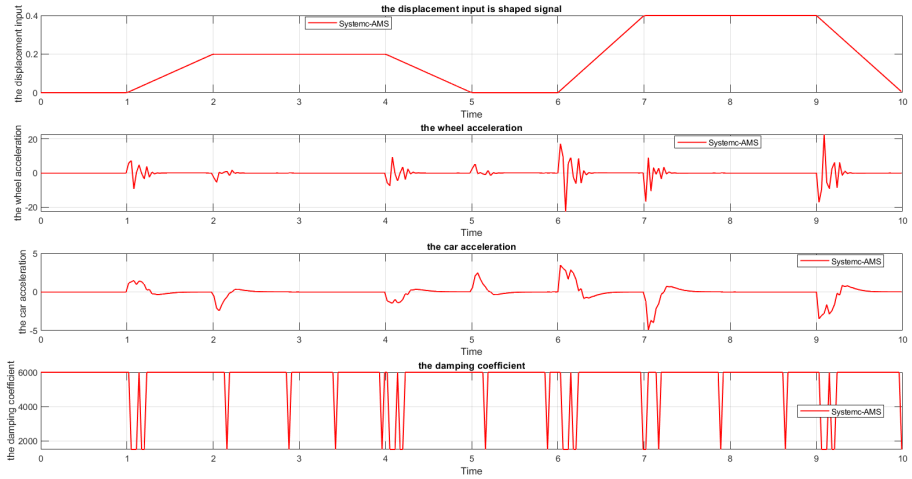

Figure 5.22: Shaped signal, step size = 30ms, simulation time = 10s, only SystemC-AMS

Applying the first configuration that reduce the step size to 10ms, the simulation results are not similar. SystemC-AMS results are much better Simulink in this case, as SystemC-AMS results are similar to the results when reducing step size to 1ms as we will see in the following words.

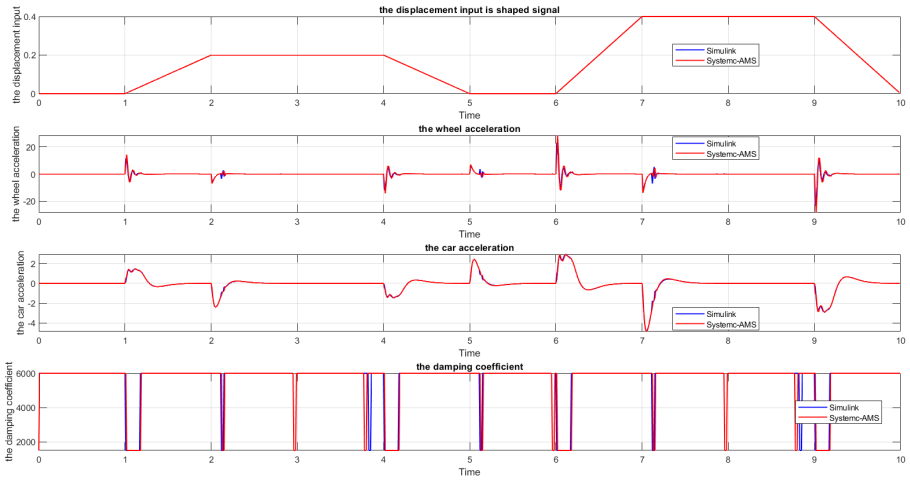

Figure 5.23: Shaped signal, step size = 10ms, simulation time = 10s

The max errors of wheel and car acceleration are large since the there is a difference in the simulation results between SystemC-AMS and Simulink.

	ERROR (%) w.r.t. SystemC-AMS		
SIMULINK SOLVER	MIN	MEAN	MAX
ODE1b (backward Euler)	0	5.1871%	90.1883%
ODE14x (extrapolation)	0	0.3900%	37.8031%
ODE1 (Euler)	0	4.2956e-05%	97.3004%
ODE2 (Heun)	0	10.3014%	45.7699%
ODE3 (Bogacki-Shampine)	0	2.7318%	37.3122%
ODE4 (Runge-Kutta)	0	3.5287%	39.0879%
ODE5 (Dormand-Prince)	0	3.5048%	38.8791%
ODE8 (Dormand-Prince)	0	3.0124%	36.8874%

Table 5.37: Wheel acceleration results, shaped signal, step size = 10ms, simulation time = 10s

	ERROR (%) w.r.t. SystemC-AMS		
SIMULINK SOLVER	MIN	MEAN	MAX
ODE1b (backward Euler)	0	0.3848%	18.2951%
ODE14x (extrapolation)	0	0.0277%	11.6004%
ODE1 (Euler)	0	0.0012%	121.5061%
ODE2 (Heun)	0	0.8336%	25.1840%
ODE3 (Bogacki-Shampine)	0	0.2035%	17.3355%
ODE4 (Runge-Kutta)	0	0.3031%	14.1937%
ODE5 (Dormand-Prince)	0	0.2720%	15.3338%
ODE8 (Dormand-Prince)	0	0.2568%	15.3154%

Table 5.38: Car acceleration results, shaped signal, step size = 10ms, simulation time = 10s

	ERROR (%) w.r.t. SystemC-AMS		
SIMULINK SOLVER	MIN	MEAN	MAX
ODE1b (backward Euler)	0%	0.8128%	75%
ODE14x (extrapolation)	0	0.0806%	75%
ODE1 (Euler)	0	0.0806%	75%
ODE2 (Heun)	0	0.8791%	75%
ODE3 (Bogacki-Shampine)	0	0.3215%	75%
ODE4 (Runge-Kutta)	0	0.3215%	75%
ODE5 (Dormand-Prince)	0	0.2413%	75%
ODE8 (Dormand-Prince)	0	0.3215%	75%

Table 5.39: Damping coefficient results, shaped signal, step size= 10ms, simulation time = 10s

Applying the second configuration, the step size to be 1ms, the simulation results now are similar.

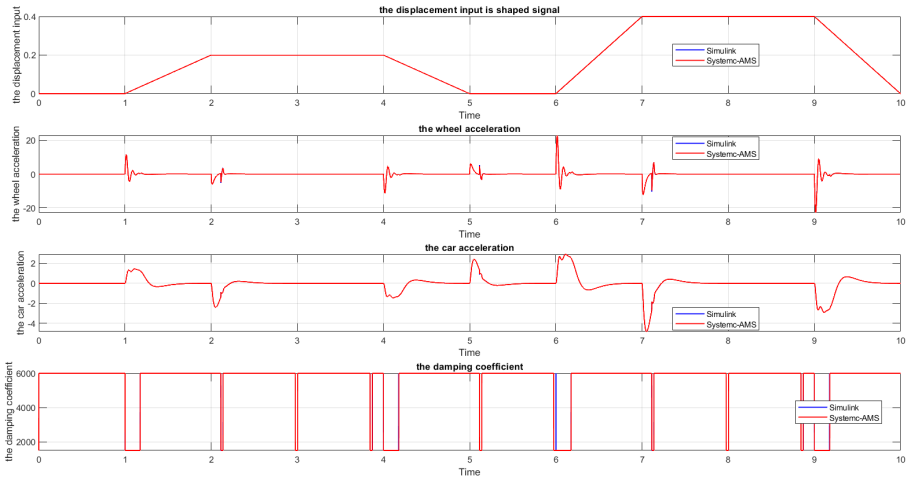

Figure 5.24: Shaped signal, step size = 1ms, simulation time = 10s

Mean and max errors of car and wheel acceleration are decreased. Compared to Runge-Kutta solver, the mean error of wheel acceleration became 0.0010% and for damping coefficient became 0.0242%.

	ERROR (%) w.r.t. SystemC-AMS		
SIMULINK SOLVER	MIN	MEAN	MAX
ODE1b (backward Euler)	0	0.2709%	8.6560%
ODE14x (extrapolation)	0	0.1297%	45.5358%
ODE1 (Euler)	0	5.4249e-05%	44.8581%
ODE2 (Heun)	0	0.0211%	45.7581%
ODE3 (Bogacki-Shampine)	0	0.0815%	44.9268%
ODE4 (Runge-Kutta)	0	0.0010%	45.7413%
ODE5 (Dormand-Prince)	0	0.0085%	45.7972%
ODE8 (Dormand-Prince)	0	0.1774%	47.5074%

Table 5.40: Wheel acceleration results, shaped signal, step size = 1ms, simulation time = 10s

	ERROR (%) w.r.t. SystemC-AMS		
SIMULINK SOLVER	MIN	MEAN	MAX
ODE1b (backward Euler)	0	0.0279%	1.8397%
ODE14x (extrapolation)	0	0.0140%	17.6600%
ODE1 (Euler)	0	2.3833e-04%	18.4979%
ODE2 (Heun)	0	3.7587e-04%	17.6661%
ODE3 (Bogacki-Shampine)	0	0.0078%	17.3967%
ODE4 (Runge-Kutta)	0	4.8483e-04%	17.6850%
ODE5 (Dormand-Prince)	0	0.0015%	17.6985%
ODE8 (Dormand-Prince)	0	0.0190%	18.3357%

Table 5.41: Car acceleration results, shaped signal, step size = 1ms, simulation time = 10s

	ERROR (%) w.r.t. SystemC-AMS		
SIMULINK SOLVER	MIN	MEAN	MAX
ODE1b (backward Euler)	0	0.0965%	75%
ODE14x (extrapolation)	0	0.0724%	75%
ODE1 (Euler)	0	0.0322%	75%
ODE2 (Heun)	0	0.0242%	75%
ODE3 (Bogacki-Shampine)	0	0.0081%	75%
ODE4 (Runge-Kutta)	0	0.0242%	75%
ODE5 (Dormand-Prince)	0	0.0322%	75%
ODE8 (Dormand-Prince)	0	0.0161%	75%

Table 5.42: Damping coefficient results, shaped signal, step size = 1ms, simulation time = 10s

By applying third configuration, step size is 0.1ms, the simulation results became identical.

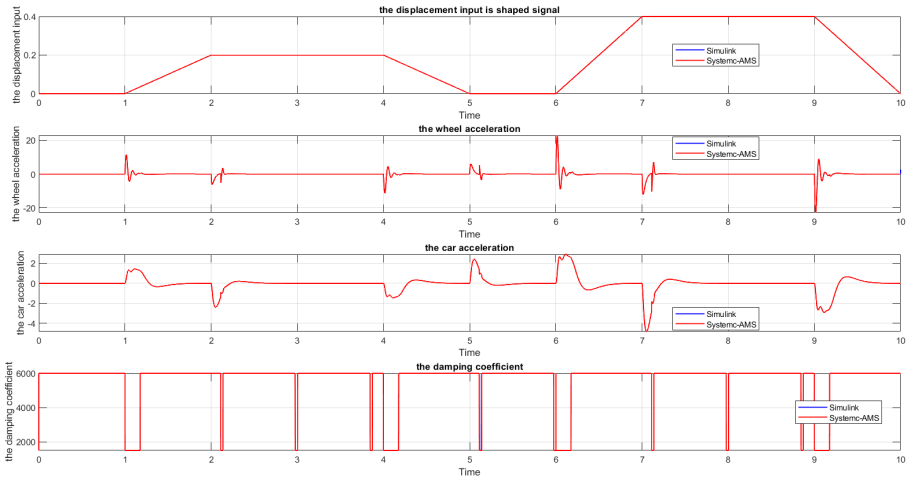

Figure 5.25: Shaped signal, step size = 0.1ms, simulation time = 10s

	ERROR (%) w.r.t. SystemC-AMS		
SIMULINK SOLVER	MIN	MEAN	MAX
ODE1b (backward Euler)	0	0.0037%	10.6269%
ODE14x (extrapolation)	0	0.0038%	48.0857%
ODE1 (Euler)	0	0.0295%	47.7508%
ODE2 (Heun)	0	0.0297%	47.8343%
ODE3 (Bogacki-Shampine)	0	0.0283%	47.8483%
ODE4 (Runge-Kutta)	0	0.0208%	47.9182%
ODE5 (Dormand-Prince)	0	0.0284%	47.8443%
ODE8 (Dormand-Prince)	0	0.0211%	47.9173%

Table 5.43: Wheel acceleration results, shaped signal, step size = 0.1ms, simulation time = 10s

	ERROR (%) w.r.t. SystemC-AMS		
SIMULINK SOLVER	MIN	MEAN	MAX
ODE1b (backward Euler)	0	0.0029%	0.4041%
ODE14x (extrapolation)	0	0.0029%	18.5671%
ODE1 (Euler)	0	1.8714e-04%	18.5555%
ODE2 (Heun)	0	1.8744e-04%	18.4704%
ODE3 (Bogacki-Shampine)	0	3.2532e-04%	18.4757%
ODE4 (Runge-Kutta)	0	0.0011%	18.5027%
ODE5 (Dormand-Prince)	0	3.1221e-04%	18.4743%
ODE8 (Dormand-Prince)	0	0.0011%	18.5023%

Table 5.44: Car acceleration results, shaped signal, step size = 0.1ms, simulation time = 10s

	ERROR (%) w.r.t. SystemC-AMS		
SIMULINK SOLVER	MIN	MEAN	MAX
ODE1b (backward Euler)	0	0.0113%	75%
ODE14x (extrapolation)	0	0.0024%	75%
ODE1 (Euler)	0	0.0056%	75%
ODE2 (Heun)	0	0.0081%	75%
ODE3 (Bogacki-Shampine)	0	0.0032%	75%
ODE4 (Runge-Kutta)	0	0.0040%	75%
ODE5 (Dormand-Prince)	0	0.0056%	75%
ODE8 (Dormand-Prince)	0	0.0032%	75%

Table 5.45: Damping coefficient results, shaped signal, step size = 0.1ms, simulation time = 10s

5.3 Comparison of simulation speed

In this part I evaluated the simulation speed between SystemC-AMS and Simulink by applying the same three types of signals for the displacement input when damping coefficient is constant and after applying Skyhook controller, and also by varying the time step from 10ms to 0.1ms. The procedures are:

- Evaluating the Execution Time of the Simulink model in all fixed step size solvers of Simulink.

- Calculating average execution time in each case.

- Evaluating the Execution Time of the final SystemC-AMS model, and comparing it with the average execution time of Simulink.

5.3.1 Constant damping coefficient

In this part we have only the plant with constant damping coefficient, by disabling the Skyhook controller.

Constant displacement signal

First applying the constant displacement input signal, and evaluating the execution time of the Simulink model. here is the table that contains the execution time of Simulink:

	STEP SIZE		
Simulink SOLVER	10ms	1ms	0.1ms
ODE1b	0.518s	1.215s	5.424s
ODE14x	0.819s	2.200s	14.923s
ODE1	0.578s	0.897s	1.810s
ODE2	0.514s	0.816s	2.068s
ODE3	0.487s	0.887s	2.513s
ODE4	0.493s	0.929s	3.645s
ODE5	0.546s	1.023s	3.653s
ODE8	0.570s	1.212s	6.073s

Table 5.46: Execution time when the displacement input is constant signal without Skyhook controller, simulation time = 3s

Then calculating average execution time of Simulink solvers and evaluating the execution time of SystemC-AMs model as shown in the following table:

	EXECUTION TIME	
STEP SIZE	SystemC-AMS	Simulink
10ms	0.039s	0.565s
1ms	0.217s	1.150s
0.1ms	1.046s	5.013s

Table 5.47: Comparing execution time of SystemC-AMS and Simulink when the displacement input is constant signal without Skyhook controller, simulation time = 3s

We can notice that, when the step size is 10ms, the execution time of SystemC-AMS is about 14.49 times faster than the average execution time of Simulink. In all cases SystemC-AMS execution time is faster than Simulink.

We can notice that, in case of constant displacement signal when the step size is 10ms, the execution time of SystemC-AMS is about 14.49 times faster than the average execution time of Simulink. In case step size is 1ms, the execution time of SystemC-AMS is about 5.29 times faster than the average execution time of Simulink. In case step size is 0.1ms, the execution time of SystemC-AMS is about 4.78 times faster than the average execution time of Simulink.

Ramp displacement signal

Then applying the ramp displacement input signal, and evaluating the execution time of the Simulink model. here is the table that contains the execution time of Simulink:

	STEP SIZE		
Simulink SOLVER	10ms	1ms	0.1ms
ODE1b	0.590s	1.234s	5.515s
ODE14x	0.892s	2.220s	15.710s
ODE1	0.516s	0.755s	1.816s
ODE2	0.533s	0.801s	2.234s
ODE3	0.592s	0.856s	2.581s
ODE4	0.497s	0.854s	3.004s
ODE5	0.529s	1.039s	3.763s
ODE8	0.554s	1.318s	6.733s

Table 5.48: Execution time when the displacement input is ramp signal without Skyhook controller, simulation time = 3s

Then calculating average execution time of Simulink solvers and evaluating the execution time of SystemC-AMs model as shown in the following table:

	EXECUTION TIME	
STEP SIZE	SystemC-AMS	Simulink
10ms	0.034s	0.587s
1ms	0.204s	1.099s
0.1ms	1.248s	5.169s

Table 5.49: Comparing execution time of SystemC-AMS and Simulink when the displacement input is ramp signal without Skyhook controller, simulation time = 3s

We can notice that, in case of ramp displacement signal when the step size is 10ms, the execution time of SystemC-AMS is about 17.26 times faster than the average execution time of Simulink. In case step size is 1ms, the execution time of SystemC-AMS is about 5.38 times faster than the average execution time of Simulink. In case step size is 0.1ms, the execution time of SystemC-AMS is about 4.14 times faster than the average execution time of Simulink.

Shaped displacement signal

Then I applied the shaped displacement input signal, and evaluating the execution time of the Simulink model. Here is the table that contains the execution time of Simulink:

	STEP SIZE		
Simulink SOLVER	10ms	1ms	0.1ms
ODE1b	0.845s	2.537s	16.126s
ODE14x	1.263s	5.559s	54.113s
ODE1	0.596s	1.056s	4.372s
ODE2	0.555s	1.270s	5.590s
ODE3	0.596s	1.425s	6.607s
ODE4	0.584s	1.536s	7.878s
ODE5	0.780s	1.775s	10.311s
ODE8	0.884s	2.676s	18.801s

Table 5.50: Execution time when the displacement input is shaped signal without Skyhook controller, simulation time = 10s

Then calculating average execution time of Simulink solvers and evaluating the execution time of SystemC-AMs model as shown in the following table:

	EXECUTION TIME	
STEP SIZE	SystemC-AMS	Simulink
10ms	0.076s	0.763s
1ms	0.868s	1.230s
0.1ms	4.697s	15.470s

Table 5.51: Comparing execution time of SystemC-AMS and Simulink when the displacement input is shaped signal without skyhook controller, simulation time = 10s

We can notice that, in case of shaped displacement signal when the step size is 10ms, the execution time of SystemC-AMS is about 10 times faster than the average execution time of Simulink. In case step size is 1ms, the execution time of SystemC-AMS is about 1.4 times faster than the average execution time of Simulink. In case step size is 0.1ms, the execution time of SystemC-AMS is about 3.29 times faster than the average execution time of Simulink.

5.3.2 Variable damping coefficient

In this part we have the suspension system: the plant plus the Skyhook controller.

Constant displacement signal

First applying the constant displacement input signal, and evaluating the execution time of the Simulink model. here is the table that contains the execution time of Simulink:

	STEP SIZE		
Simulink SOLVER	10ms	1ms	0.1ms
ODE1b	0.547s	1.204s	5.448s
ODE14x	0.936s	2.252s	14.899s
ODE1	0.459s	0.742s	1.695s
ODE2	0.583s	0.798s	2.139s
ODE3	0.456s	0.859s	2.481s
ODE4	0.549s	0.866s	2.881s
ODE5	0.532s	0.966s	3.572s
ODE8	0.579s	1.374s	6.400s

Table 5.52: Execution time when the displacement input is constant signal and by applying Skyhook controller, simulation time = 3s

Then calculating average execution time of Simulink solvers and evaluating the execution time of SystemC-AMs model as shown in the following table:

	EXECUTION TIME	
STEP SIZE	SystemC-AMS	Simulink
10ms	0.049s	0.663s
1ms	0.180s	1.133s
0.1ms	1.290s	4.940s

Table 5.53: Comparing execution time of SystemC-AMS and Simulink when the displacement input is constant signal and by applying Skyhook controller, simulation time = 3s

We can notice that, in case of constant displacement signal when the step size is 10ms, the execution time of SystemC-AMS is about 13.53 times faster than the average execution time of Simulink. In case step size is 1ms, the execution time of SystemC-AMS is about 6.29 times faster than the average execution time of Simulink. In case step size is 0.1ms, the execution time of SystemC-AMS is about 3.83 times faster than the average execution time of Simulink.

Ramp displacement signal

Then applying the ramp displacement input signal, and evaluating the execution time of the Simulink model. here is the table that contains the execution time of Simulink:

	STEP SIZE		
Simulink SOLVER	10ms	1ms	0.1ms
ODE1b	0.638s	1.252s	5.418s
ODE14x	0.900s	2.253s	15.762s
ODE1	0.495s	0.839s	1.804s
ODE2	0.543s	0.853s	2.215s
ODE3	0.476s	0.861s	2.576s
ODE4	0.575s	0.860s	3.097s
ODE5	0.545s	0.996s	3.705s
ODE8	0.558s	1.443s	7.499s

Table 5.54: Execution time when the displacement input is ramp signal and by applying Skyhook controller, simulation time = 3s

Then calculating average execution time of Simulink solvers and evaluating the execution time of SystemC-AMs model as shown in the following table:

	EXECUTION TIME	
STEP SIZE	SystemC-AMS	Simulink
10ms	0.040s	0.590s
1ms	0.240s	1.170s
0.1ms	1.770s	4.320s

Table 5.55: Comparing execution time of SystemC-AMS and Simulink when the displacement input is ramp signal and by applying Skyhook controller, simulation time = 3s

We can notice that, in case of ramp displacement signal when the step size is 10ms, the execution time of SystemC-AMS is about 14.75 times faster than the average execution time of Simulink. In case step size is 1ms, the execution time of SystemC-AMS is about 4.875 times faster than the average execution time of Simulink. In case step size is 0.1ms, the execution time of SystemC-AMS is about 2.44 times faster than the average execution time of Simulink.

Shaped displacement signal

Then applying the shaped displacement input signal, and evaluating the execution time of the Simulink model. here is the table that contains the execution time of Simulink:

	STEP SIZE		
Simulink SOLVER	10ms	1ms	0.1ms
ODE1b	0.817s	2.316s	16.881s
ODE14x	1.268s	6.643s	50.994s
ODE1	0.531s	1.213s	4.206s
ODE2	0.551s	1.295s	5.513s
ODE3	0.589s	1.414s	6.651s
ODE4	0.593s	1.755s	7.770s
ODE5	0.780s	1.751s	11.223s
ODE8	0.899s	2.654s	19.122s

Table 5.56: Execution time when the displacement input is shaped signal and by applying Skyhook controller, simulation time = 10s

Then calculating average execution time of Simulink solvers and evaluating the execution time of SystemC-AMs model as shown in the following table:

	EXECUTION TIME	
STEP SIZE	SystemC-AMS	Simulink
10ms	0.067s	0.754s
1ms	0.933s	2.380s
0.1ms	4.99s	15.295s

Table 5.57: Comparing execution time of SystemC-AMS and Simulink when the displacement input is shaped signal and by applying Skyhook controller, simulation time = 10s

We can notice that, in case of shaped displacement signal when the step size is 10ms, the execution time of SystemC-AMS is about 11.25 times faster than the average execution time of Simulink. In case step size is 1ms, the execution time of SystemC-AMS is about 2.55 times faster than the average execution time of Simulink. In case step size is 0.1ms, the execution time of SystemC-AMS is about 3 times faster than the average execution time of Simulink. As conclusion, SystemC-AMS simulation speed is faster than the case of Simulink, still achieving a good accuracy.

Chapter 6

Conclusions

The thesis focused on modeling and simulating (Model-in-the-Loop) an automotive system using SystemC-AMS instead of Simulink. The thesis proved the feasibility by modeling a suspension system in both frameworks. This highlighted that both tools achieve similar accuracy up to 99.99% in several simulation configurations, despite of the different solvers and scheduling semantics. At the same time, SystemC-AMS exhibits significantly faster simulation speed, achieving a speedup of up to 17.26 times compared to Simulink. Modeling in Simulink is more user-friendly, both in terms of graphical support and of available blocks, e.g.:

- the Add block which can be handled in order to have two or more input ports, depending on the number of inputs to be added, while SystemC-AMS offers similar block that can add only two signals;
- Simulink offers a Discrete-Time Integrator that does not exists in SystemC-AMS;
- for the generation of the input source, in Simulink there is a Signal Builder block for designing complex input wave-forms and defining their characteristics, commonly used for tasks such as generating test signals. On contrary, in SystemC-AMS, the user have to describe the input signals manually by writing some codes that describe the required signals;
- In SystemC-AMS, in order to model a dynamic systems that uses TDF it is necessary to add a Delay block in order to have a scheduled system. In contrary, there is no need to add a Delay block in Simulink, that natively handles dynamic systems;
- Simulink provides a user-friendly GUI, while SystemC-AMS code has to be written manually.

In future work, I will investigate the adoption of the COSIDE tool by COSEDA, that provides a GUI-based design environment [3], and apply the same methodology to other automotive components, to further deepen the analysis.

Bibliography

[1] Amal Banerjee and Balmiki Sur. *SystemC and SystemC-AMS in Practice: SystemC 2.3, 2.2 and SystemC-AMS 1.0*. Springer Science & Business Media, 2013.

[2] M Bruqi, R Likaj, and A Shala. «Simulation of vertical quarter car model with one and two DOFs». In: *Machines. Technologies. Materials.* 11.6 (2017), pp. 261–263.

[3] COSEDA Technologies GmbH. *COSIDE - The Design Environment for Heterogeneous Systems*. https://www.coseda-tech.com/. 2023.

[4] Accellera Systems Initiative et al. *Standard SystemC® AMS extensions 2.0 Language Reference Manual*. 2016.

[5] Accellera Systems Initiative et al. *SystemC AMS extensions Users Guide*. 2020.

[6] Saad Bin Abul Kashem et al. «Modeling and simulation of electromagnetic damper to improve performance of a vehicle during cornering». PhD thesis. Swinburne University of Technology, 2013.

[7] Wenjun Li and Hongkun Zhang. «Embedded System Design for Vehicle Semi-active Suspension System». In: *2016 5th International Conference on Energy and Environmental Protection (ICEEP 2016)*. Atlantis Press. 2016, pp. 238–243.