An Investigation of Model-Based Techniques for Automotive Electronic System Development

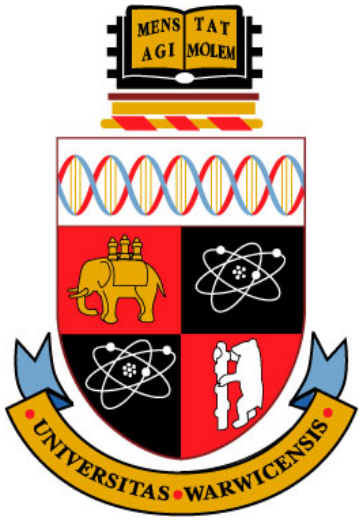
by

Yue Guo

A thesis submitted in partial fulfillment of the requirements for the degree of Doctor of Philosophy in Engineering

School of Engineering

University of Warwick



September 2009

Contents

List of Tables

List of Figures

Acknowledgements

First and foremost, I would like to express my sincere gratitude to my supervisor, Dr. R. Peter Jones. I am indebted to him for the opportunity to conduct research at the University of Warwick. Without his consistent and illuminating instructions, this thesis could not have reached its present form. His keen and vigorous academic observation enlightens me not only in this thesis but also in my future study.

I am very grateful to Mr. Ross McMurran for allowing me to be involved in the research project at the International Automotive Research Centre (IARC) and providing me with useful knowledge of in-vehicle infotainment system.

I would like to express my heartfelt gratitude to Dr. Arun Chakrapani Rao who has provided me with valuable knowledge in system modelling. He was particularly gracious and responsive when I encountered problems or had questions.

Sincere gratitude also goes to my colleagues at IARC: Mark Amor-Segan, Gunny Dhadyalla, Dr. Yingping Huang, Dr. Jittiwut Suwatthikul, Dr. Suguna Thanagasundram, Dr. Bo Wang, Dr. Caizhen Cheng; the former IARC members: Dr. David Antory and Roopa Arun.

I appreciate the contribution to this thesis made in various ways by my wonderful friends, in particular: Yue Li, Yupeng Wu, Xiang Yuan and Xiangyu Kong.

Special thanks go to my beautiful wife Nan Peng for her love, encouragement and great confidence in me all through these years.

As always, the people who matter most are left until last – my beloved parents, Yun Song and Chengzhu Guo. My deepest gratitude goes to my parents for their endless and unconditional love. I will always love you.

Yue Guo

September, 2009.

Declarations

I hereby declare that the material in this thesis has not been submitted for a higher degree at any other university. This thesis entirely contains research work carried out by Yue Guo under the supervision of Dr. R. Peter Jones, unless references are given.

Some parts of the following chapters were submitted for publication:

Chapter 3: Yue Guo, Arun Chakrapani Rao and R. Peter Jones, "Architectural and Functional Modelling of an Automotive Driver Information System Using SysML," in *Proceedings of 2008 IEEE/ASME International Conference on Mechatronic and Embedded Systems and Applications*, pp. 552-557, Beijing, China, October. 2008.

Chapter 4: Yue Guo, Arun Chakrapani Rao and R. Peter Jones, "Architectural and Functional Modelling of an Automotive Driver Information System Using SysML," in *Proceedings of 2008 IEEE/ASME International Conference on Mechatronic and Embedded Systems and Applications*, pp. 552-557, Beijing, China, October. 2008;

Yue Guo and R. Peter Jones, "A Study of Approaches for Model Based Development of an Automotive Driver Information System," in *Proceedings of 2009 IEEE International Systems*

Conference, pp. 267-272, Vancouver, British Columbia, Canada, March. 2009.

Chapter 5: Yue Guo and R. Peter Jones, "A Study of Approaches for Model Based Development of an Automotive Driver Information System," in *Proceedings of 2009 IEEE International Systems Conference,* pp. 267-272, Vancouver, British Columbia, Canada, March. 2009.

Abstract

Over the past decades, the adoption of electronic systems for the manufacturing of automotive vehicles has been exponentially popularized. This growth has been driven by the premium automobile sector where, presently, diverse electronic systems are used. These electronic systems include systems that control the engine, transmission, suspension and handling of a vehicle; air bag and other advanced restraint systems; comfort systems; security systems; entertainment and information (infotainment) systems. In systems terms, automotive embedded electronic systems can now be classified as a System of Systems (SoS). Automotive systems engineering requires a sustainable integration of new methods, development processes, and tools that are specifically adapted to the automotive domain. Model-based design is one potential methodology to carry out design, implement and manage such complex distributed systems, and their integration into one cohesive and reliable SoS to meet the challenges for the automotive industry.

This research was conducted to investigate the model-based design of a 4×4 Information System, within an automotive electronic SoS. Two distinct model-based approaches to the development of an automotive electronic system are discussed in this study. The first approach involves the use of the Systems Modelling Language (SysML) based tool ARTiSAN Studio for structural modelling, functional modelling and code generation. The second approach

involves the use of the MATLAB based tools Simulink and Stateflow for functional modelling, and code generation. The results show that building the model in SysML by using ARTiSAN Studio provides a clearly structured visualization of the 4×4 Information System from both structural and behavioural viewpoints of the system with relevant objects. SysML model facilitates a more comprehensive understanding of the system than the model built in Simulink/Stateflow. The Simulink/Stateflow model demonstrates its superior performance in producing high quality and better efficiency of C code for the automotive software delivery compared with the model built in ARTiSAN Studio. Furthermore, this Thesis also gets insight into an advanced function development approach based on the real-time simulation and animation for the 4×4 Information System. Finally, the Thesis draws conclusions about how to make use of model-based design for the development of an automotive electronic SoS.

Abbreviations

ABS	Antilock Braking System
CAN	Controller Area Network
ECU	Electronic Control Unit
HDC	Hill Descent Control
HEV	Hybrid Electric Vehicle
HLDF	High Level Display Front
HMI	Human Machine Interface
LCD	Liquid Crystal Display
MOST	Media Oriented System Transport
OMG	Object Management Group
RTE	Run Time Error
RTW	Real-Time Workshop
RTW EC	Real-Time Workshop Embedded Coder
SE	System Engineering
SoS	System of Systems
SoSE	System of Systems Engineering
SysML	Systems Modelling Language
TO	Terrain Optimization
UML	Unified Modelling Language

Chapter 1

Introduction

In the late 1880s, the first automobile was built in Mannheim, Germany, by Karl Benz. In 1908, the Ford Model T which is generally regarded as the first affordable automobile was built. In these early stages of the automotive industry, the use of electrical systems on the vehicle was very limited, supporting ignition and lighting only [1]. Those vehicles provided minimal driver information through an analogue display. For example, the Ford Model T only included basic instruments such as mechanical speedometer, engine temperature and fuel level indicator as the information system. There was no entertainment system at that time.

The first practical car radio was believed to be invented in the early 1920s by William Lear. It was the first and only entertainment system on the vehicle during that period. In the 1960s, the tape player was installed and the CD player was first introduced in 1984 [2].

The electrical system evolved slowly until the microprocessor was introduced in 1971. One of the first microprocessor applications in cars was an advanced ignition system built by Delco-Remy for the 1977 Oldsmobile Toronado [3]. As part of electronic control unit (ECU), the microprocessor can process more

information and display it to the driver. The ECU was used for engine management for the first time at that period. Other applications of ECUs in vehicles included transmission-shift control, Antilock Braking System (ABS) and instrument cluster as shown in Fig. 1-1. However, in the early days of automotive electronics, each new function was implemented as a stand-alone ECU which is a subsystem composed of a microprocessor, memory, input and output [4].

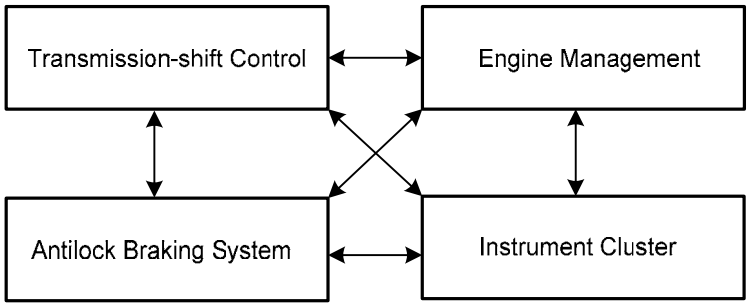


Fig. 1-1. Conventional data transfer.

As shown in Fig. 1-1, data exchanges through point-to-point links between ECUs. This requires a large number of wires and therefore soon reached its practical limits. Besides, the amount of associated connectors is very difficult to manage [5, 6].

In response to these practical limits, the industry moved from point-to-point data communication to data bus technology. In the mid-1980s, Bosch developed the Controller Area Network (CAN), one of the first and most enduring automotive control networks [7]. CAN is a communication technique that consists of a twisted pair of copper wires in which data is transmitted on a special network as shown schematically in Fig. 1-2 [1]. Far fewer connections are needed by using this CAN network on the vehicle. The room saved can be used to accommodate more sensors, actuators and ECUs. As a result, more vehicle information can be processed and delivered to the driver.

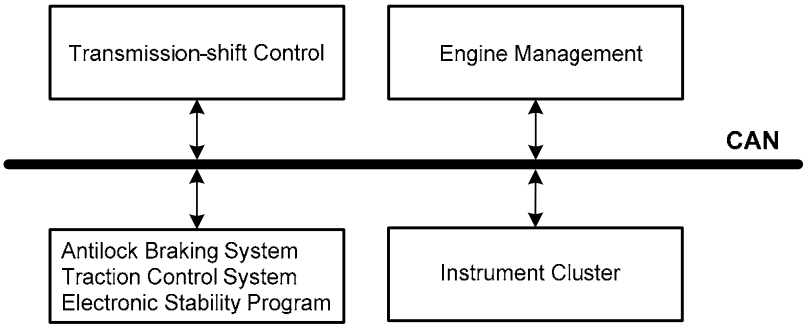


Fig. 1-2. CAN bus topology.

Later in the 1990s, a new network protocol called MOST was introduced in the automotive vehicles [8]. MOST is the acronym of Media Oriented Systems Transport. It is a fiber-optic network protocol with capacity for high-volume streaming. It is designed for multimedia applications in the automotive environment. MOST gives the advantages of ease of use, cost effectiveness and flexibility. It supports real-time and high volume data transmission such as data from a navigation system and from the DVD player which made its first appearance in vehicles in the late 1990s.

As a result of the emergence of electronic control and networking technologies, the past 30 years have witnessed a near exponential growth of in-vehicle, embedded electronic systems as shown in Fig. 1-3 [9]. This growth has been driven by the premium automobile sector where, presently, electronics and software account for around 40% of the value of some vehicles [10]. Current in-vehicle electronic systems are diverse and include: systems that control the engine, transmission, suspension and handling of a vehicle; air bag and other advanced restraint systems; comfort systems; security systems; entertainment and information (infotainment) systems [4]. Such wide ranging functionality is enabled by networks of up to 50, or more, ECUs that are distributed throughout a vehicle. Individual ECUs host software that is required to interact with devices such as

sensors and actuators, and other ECUs, within time constraints. The ECUs are linked by a communication network, consisting of several data bus technologies that provide transmission rates which have also been subject to near exponential growth. In systems terms, automotive, embedded electronic systems can now be classified as a System of Systems (SoS) [11]. SoS means large-scale concurrent and distributed systems the components of which are systems themselves [12]. A detailed description of definition and characteristics of SoS will be given in Chapter 2.

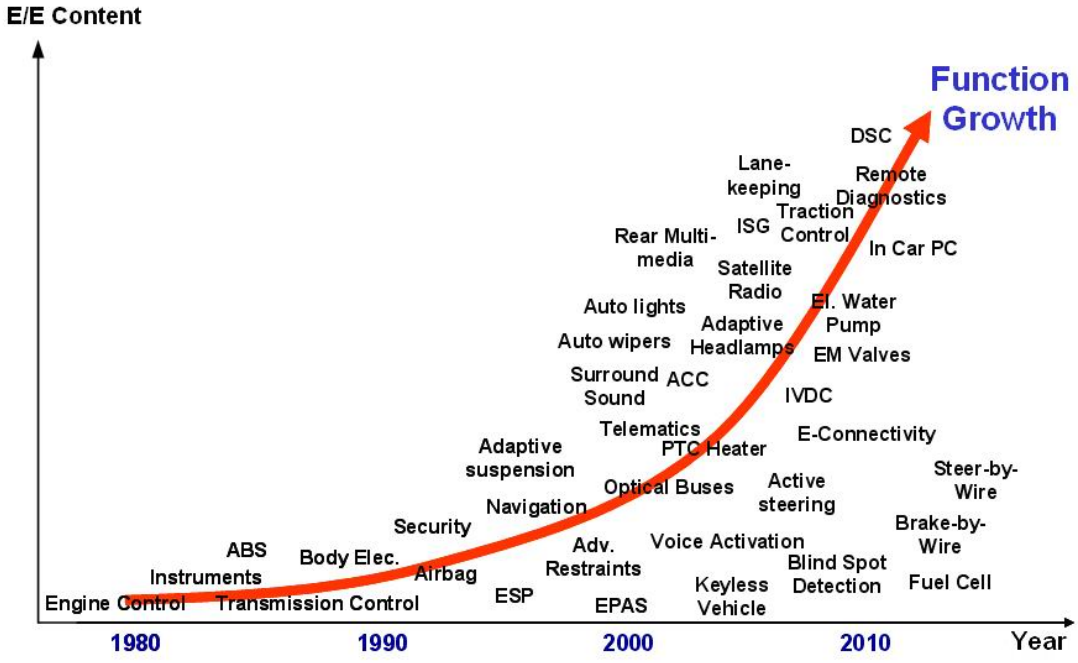


Fig. 1-3. Evolution of vehicle electrical/electronic features (adapted from [11]).

The design, implementation and management of such complex distributed systems, and their integration into one cohesive and reliable SoS are presenting new challenges for the automotive industry. To meet these challenges, it is necessary to develop new methodologies for capturing the requirements for the SoS, at the outset of the product development process, and conveying the requirements

through the stages in the product development process. Model-based design is one such potential methodology [13-16].

At present, model-based design is increasingly replacing system specification in plain text form. Such a model precisely formulates the specification documents and avoids interpretation leeway. Most importantly, the reliability and functionality of automobiles are largely dependent on software and electronic applications in recent years [17]. Such a model is unambiguous because it clearly defines the structure and functionality of the system or SoS by using advanced modelling techniques such as various modelling languages and tools [18]. It helps the engineer to gain the understanding of the system through a graphical visualization before the development. Thus, the entire automotive electronic system can be built up optimally. Moreover, auto coding is the trend of automotive software development. The development cycle has been reduced by more than half over the past two decades which has benefited from this technique [19, 20]. Various types of code can be generated from the model through the proper tools. Therefore, such a model can also shorten the development cycle from the code generation aspect.

The Unified Modelling Language (UML) [21] is a modelling language which has been playing an important role in software engineering. It has the potential to support innovative SoS modelling which ties the architecture, design and verification aspects in a unified perspective [22]. However, there are some problems and challenges with UML, such as syntactic and semantic overlap, and immature constructs [23, 24]. In order to overcome these challenges and enable it to handle the system engineering, the Systems Modelling Language (SysML) [25] is adapted from UML. The final SysML specification was released in April 2007 by the Object Management Group (OMG) – the US-based industry standards body

that manages and configures the SysML. SysML is defined on the website of the OMG as “a general purpose graphical modelling language for specifying, analyzing, designing and verifying complex systems that may include hardware, software, information, personnel, procedures and facilities” [26]. SysML is intended to unify the various modelling languages currently used by systems engineers in a similar manner to how UML unifies the modelling language used in the software industry. SysML extends the application of UML to systems which are not purely software based, and can in particular be applied to design heterogeneous embedded systems and SoS [18, 27].

At present, system engineers use a wide range of modelling languages, tools and techniques such as MATLAB/Simulink [28] which is a well known modelling and simulation environment. MATLAB is used in a wide range of applications, including signal and image processing, communications, control design, test and measurement. Simulink which is integrated with MATLAB provides an environment for modelling, simulating and analyzing multi-domain dynamic systems. In particular, they can be used for model-based design for control systems. Coupled with the Real-Time Workshop, Simulink facilitates the automatic code generation for real-time implementation of embedded systems. Stateflow, the other product developed by the MathWorks extends Simulink with a design environment for developing event-driven systems that contain control and supervisory logic.

This Thesis is going to explore model-based design. SysML based tool ARTiSAN Studio [29] will be investigated for structural modelling. Both ARTiSAN Studio and MathWorks Simulink/Stateflow will be explored and compared for functional modelling. Their capacity for the code generation will also be examined. The static analysis tool, PolySpace [30] is utilized to perform

automatic code verification for the C code generated from both ARTiSAN Studio and Simulink/Stateflow.

The research reported in this Thesis will be conducted via a case study involving the model-based design of a "4×4 Information System", which is incorporated into the infotainment system installed in high-end premium Land Rover vehicles.

Having investigated the model-based design of the 4×4 Information System, the ability to easily construct a real-time animation of the system from the automatically generated C code is examined by using dSPACE ControlDesk [31]. The Thesis presents the outcome of this research and draws conclusions about how to make use of model-based design for the development of an automotive electronic SoS.

This Thesis is structured as follows:

Chapter 2 provides background literature on the System and SoS, System Engineering (SE) and System of Systems Engineering (SoSE), automotive system development process, and the modelling languages used in this research.

Chapter 3 presents an overview of the Driver Information System and a detailed description of its 4×4 Information System chosen as the pilot study for this Thesis.

Chapter 4 discusses two distinct model-based approaches to automotive electronic system development. The first approach involves the use of the SysML based tool ARTiSAN Studio for structural modelling, functional modelling and code generation. The second approach involves the use of the MATLAB based tools Simulink/Stateflow for functional modelling, and code generation. In this chapter, the advantages and disadvantages of two approaches for the development

of an automotive electronic SoS are explored and compared. Conclusions are drawn on how to make use of the model-based design to meet the challenges in the automotive industry.

Chapter 5 demonstrates the coding implementation through both approaches in order to further investigate functional modelling. The attention focuses on the comparison of quality and efficiency of the code.

Chapter 6 explores the real-time simulation and animation of the 4×4 Information System interface by using dSPACE ControlDesk and the C code which is generated from the Simulink/Stateflow model.

Chapter 7 provides the conclusion of the Thesis and discusses the future work.

Chapter 2

Literature review

In this chapter, the literature of System, SoS, SE and SoSE are reviewed. The automotive electronic system development process is discussed with current and foreseeable challenges. New technologies to address these challenges are also discussed.

2.1 System

2.1.1 Concept of System

In this section, a detailed account of various definitions of System is given from published literature in various domains. How Systems have been clarified in various domains is detailed.

Various definitions have been used for systems. In [32], a system has been defined as a set of interrelated elements working together for some purpose. Examples of systems can be seen in various domains, such as a biological system, a management system and an automotive powertrain system.

In biology, a system could be a group of organs that work together to perform some function, such as the digestive system. All these systems have inputs, outputs, and maintain a basic level of equilibrium.

In business, a business component system is a set of cooperating business components to deliver a solution to a business problem, for example, an invoice management system or a payroll system.

In science, a system is a group of interacting, interrelated or interdependent elements forming a complex whole. An ecosystem is an example of a system in science.

The element is the basic component of a system. A system element can be either physical or conceptual [33]. It is irreducible, i.e., it can not be made by the other elements. An element which has no relationship with any other element of the system is not recognised as a part of that system. The components of a system are as shown below in Fig. 2-1.

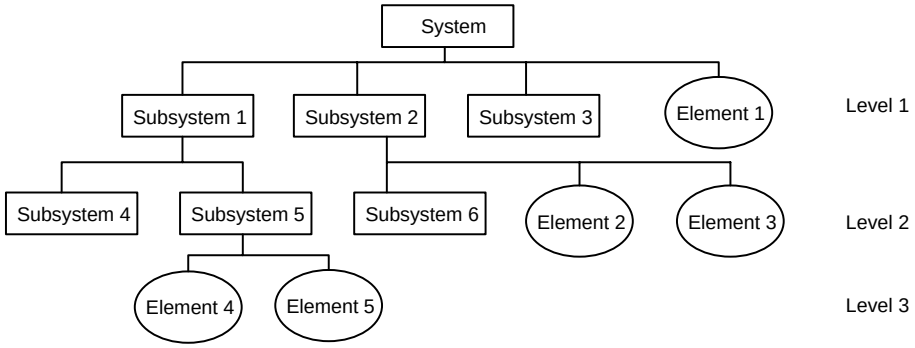


Fig. 2-1. A representative for the system structure.

A system could consist of one or more subsystems. Subsystems could be made up by the lower level subsystem or elements which is the basic component of a system [34]. The System can be viewed at different levels as shown in Fig. 2-1.

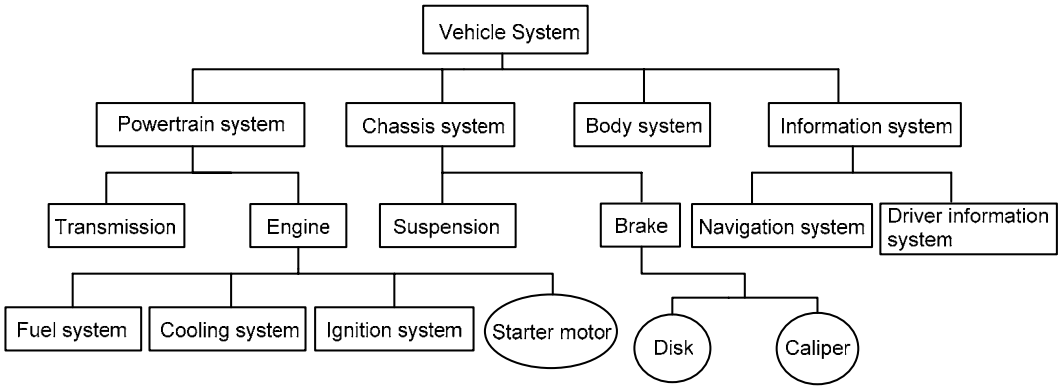


Fig. 2-2. Vehicle system.

Fig. 2-2 is an example vehicle system. This system consists of four subsystems which are the Powertrain System, the Chassis System, the Body System and the Information System. Suspension and brake are subsystems of the chassis system. A brake system has two elements which are disc and caliper.

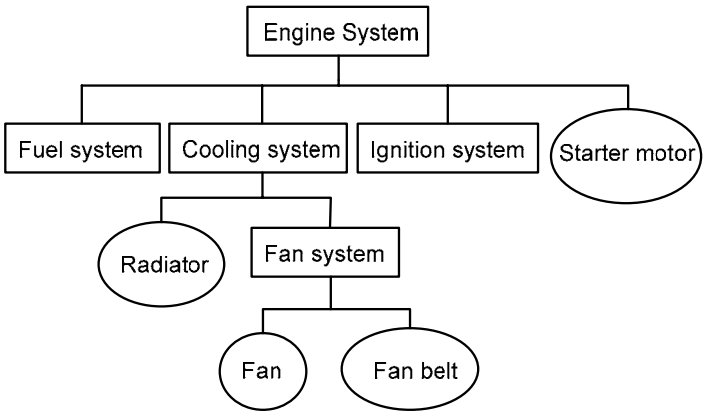


Fig. 2-3. Engine system.

A cooling system is one of the subsystems in the engine system as shown in Fig. 2-2. It can be viewed at a more detailed level as shown in Fig. 2-3. In this diagram, the cooling system is a subsystem of the engine system and it has the fan system and radiator as its component. Therefore, it can be seen that system is a relative concept. It can be a small system consisting of just one component or a large system with several subsystems.

Systems can be classified in many different ways. From the viewpoint of their basic properties, systems can be divided into static or dynamic, linear or nonlinear, continuous or discrete and so on.

2.1.2 Characteristics of a System

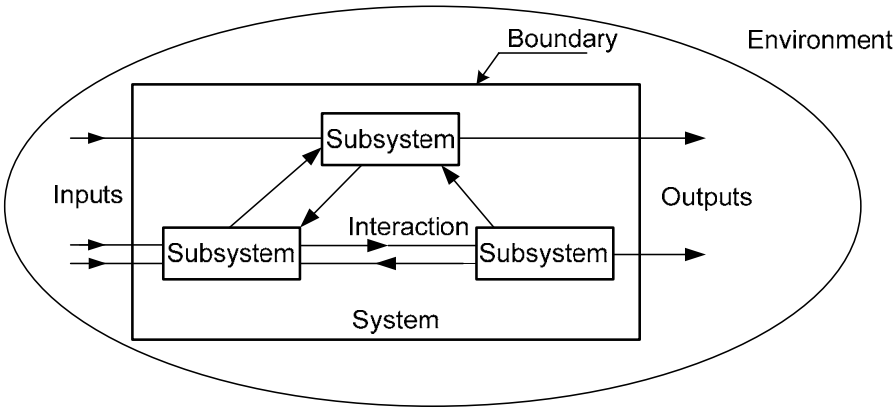


Fig. 2-4. A general depiction of a system.

A general depiction of a system is as shown in Fig. 2-4. The characteristics of a system are shown in this diagram. The subsystem is a set of elements, which is a system itself, and a part of the whole system. Each subsystem has its own function. Therefore, interaction within each subsystem should be stronger than with other subsystems. A system exists within an environment. It has a boundary separating itself from the external disturbance within its environment [33].

Every system interacts with its environment through two groups of interactions. The first one originates outside the system and does not depend on what happens in the system directly. This group of interactions is called the inputs to the system. The other group of interactions is generated by the system. This group of interactions is called outputs of the system. Output is the way by which systems affect the environment. A system returns output to its environment as a result of its

functioning [35]. Each system has a certain input and output. Systems receive inputs and generate outputs [33].

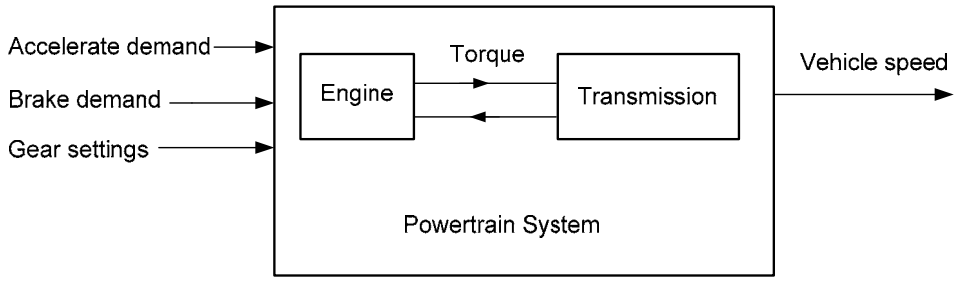


Fig. 2-5. Powertrain System elements.

Fig. 2-5 is an example showing the characteristics of the Powertrain System. Accelerate demand, brake demand and gear settings are the inputs. Engine and transmission are the subsystems and they interact through the torque change. Vehicle speed is the output of this system.

2.1.3 Discussion

Visualising a set of elements and their interrelationships as a system allows engineers look into the essential characteristics of a specific situation. Engineers study general properties of systems by emphasizing the system's inputs and outputs to exclude of external disturbance and all other details [33]. Nowadays, a system is becoming more and more complex. An example of Driver Information System is shown in Fig. 2-6.

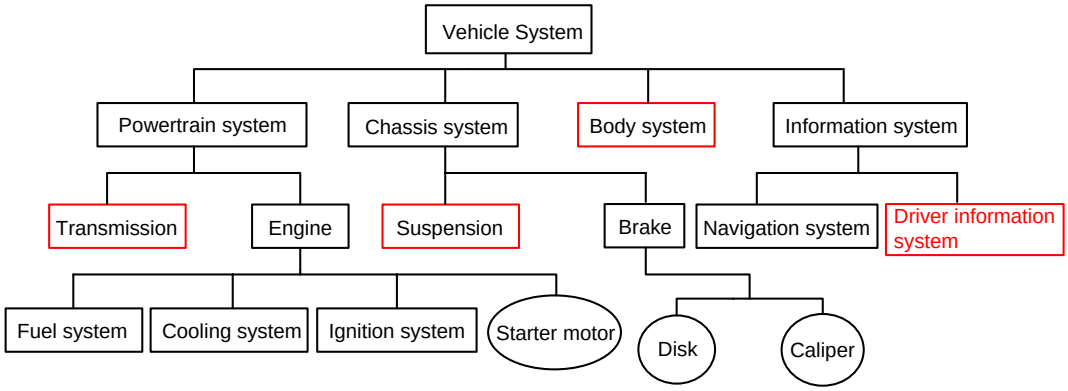


Fig. 2-6. Automotive Driver Information System.

The Driver Information System on the modern vehicle provides the driver with the ability to view information and status relating to the Powertrain System, the Chassis System and the Body System. For example, as shown in Fig. 2-6, the Driver Information System can display the gear selection and suspension height to the driver. A large amount of interaction and data exchange are needed to facilitate such advanced functions. The data delivery requires additional connection among these subsystems on the vehicle. Consequently, the automotive electronic system quickly expands to the very large or super system. The conventional approach to realize the automotive system as a system to carry out analysis and design activities is no longer suitable for very large modern automotive electronic systems. Therefore, in order to look into the causality and interrelationship of a large system or super system, developers investigating it as a SoS is discussed in the next section.

2.2 System of Systems

This section provides a detailed description of definition and characteristics of SoS.

2.2.1 Concept of System of Systems

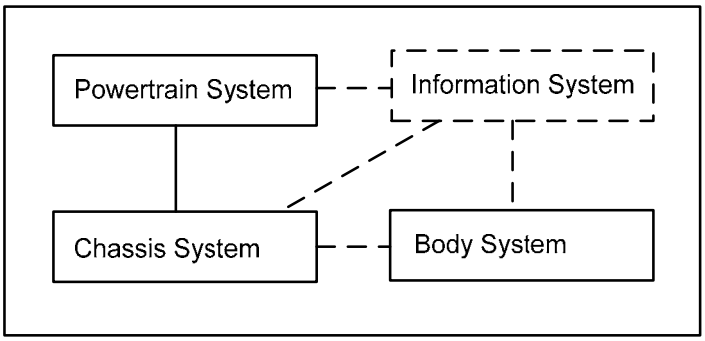


Fig. 2-7. Automotive System of Systems.

As shown in Fig. 2-7, an automotive system is presented as a typical SoS. It contains four main systems which are Powertrain System, Chassis System,

Information System and Body System. Each system can work independently and has its own functions. They also connect together as a vehicle in order to realize some higher level functions which each system can not achieve alone. Some connections between the systems are weak whilst the connection is strong between the subsystems and elements inside the system. These connections can be changed, added or removed without affecting the function of the whole SoS. For example, with the development of the automotive industry, an airbag is a standard piece of equipment in the vehicle and it is a component of the Body System. It requires the signal from the Chassis System for deploying the airbag. Therefore, the Chassis System and the Body System have to collaborate in order to enable the correct function of the airbag. It is an example that shows systems are required to be integrated as an automotive SoS to deliver some advanced functions.

2.2.2 Characteristics of System of Systems

Five different characteristics have been proposed in [12] to distinguish a SoS from a System. They are autonomy, belonging, connectivity, diversity and emergence.

2.2.2.1 Autonomy

The systems in a SoS are integrated and collaborate to achieve the goal of the SoS. Systems within a SoS have individual functions and a level of autonomy. As shown in Fig. 2-7, a typical automotive SoS consists of four systems which are the Powertrain System, Chassis System, Information System and Body System. Each system has independent functions. For example, the Powertrain System has certain inputs such as accelerate demand, brake demand and gear settings and outputs such as the vehicle speed and so on. It has the ability to work and maintain its own functions independently.

In contrast, within a system, there is little or no autonomy for subsystems or elements of the system. Therefore, autonomy is one of the characteristics of SoS.

2.2.2.2 Belonging

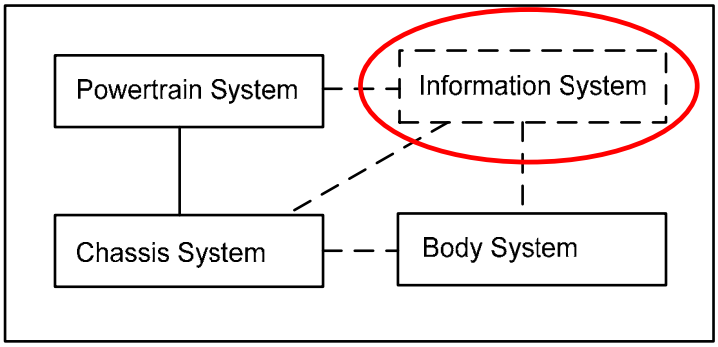


Fig. 2-8. Belonging of automotive System of Systems.

As shown in Fig. 2-8, belonging is another characteristic of SoS. Developers choose what systems to belong to a SoS. Specifically, new systems can be added into a SoS whilst one or more systems can be removed from a SoS without affecting the function of the whole SoS. Some of the latest systems in the vehicle such as the Information System and its various subsystems like navigation have been gradually integrated into a vehicle. This means that they could either be present in a vehicle or not.

2.2.2.3 Connectivity

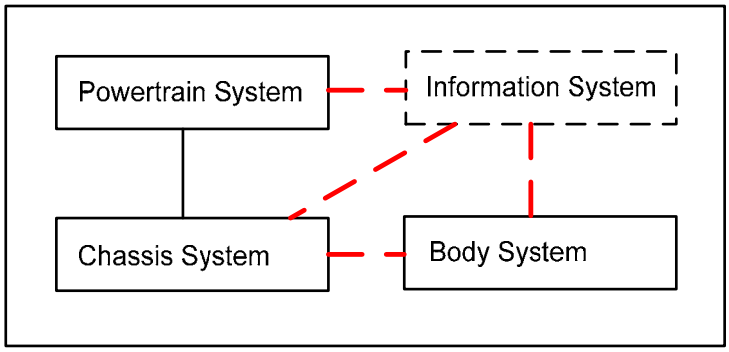


Fig. 2-9. Connectivity of automotive System of Systems.

Some connections between systems in the SoS are weak. They can be added, removed or changed. In Fig. 2-9, dashed lines show some connections between systems in the automotive SoS. These connections can be enabled or disabled at any time depending on the different requirements. To enable these connections, more information can be delivered and used between different systems.

For example, after enabling the connection between the Powertrain System and the Information System, the engine speed and gear setting can be viewed from the Information System. The suspension information like vehicle height can be displayed through the connection between the Chassis System and the Information System. The central locking information can be displayed as well if connection is enabled between the Body System and the Information System. Furthermore, in order to enable certain functions several systems need to be connected together. For example, if people do not wear the seat belt when they start the engine and drive, some models of vehicle give a warning flash or sound. It means this function needs at least the Powertrain System, the Body System and the Information System to collaborate together in order to enable this function. Besides, these connections can be removed if people choose not to view this information in the Information System and it will not affect the function of automotive system as a SoS.

2.2.2.4 Diversity

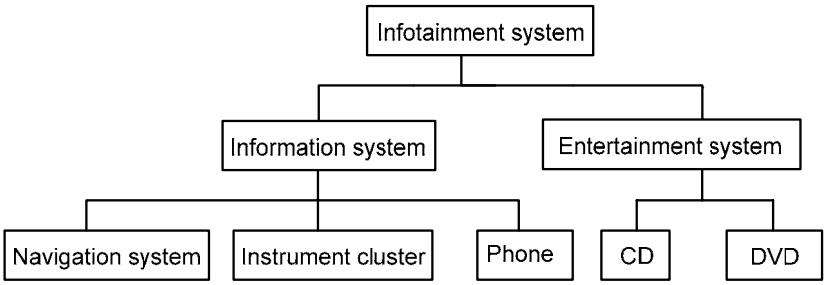


Fig. 2-10. Diversity of automotive System of Systems.

The function of a system is usually very limited but the SoS can satisfy several different requirements at the same time and present various functions. The automotive Infotainment System shown in Fig. 2-10 is such an example. This SoS not only displays some necessary information like engine speed, temperature and fuel level but also provides navigation information, radio, CD and DVD which make the Information System merge with the Entertainment System. Within an automotive SoS, information can be delivered and used between different systems to enable many advanced functions which satisfy various requirements from safety, performance, comfort, etc.

2.2.2.5 Emergence

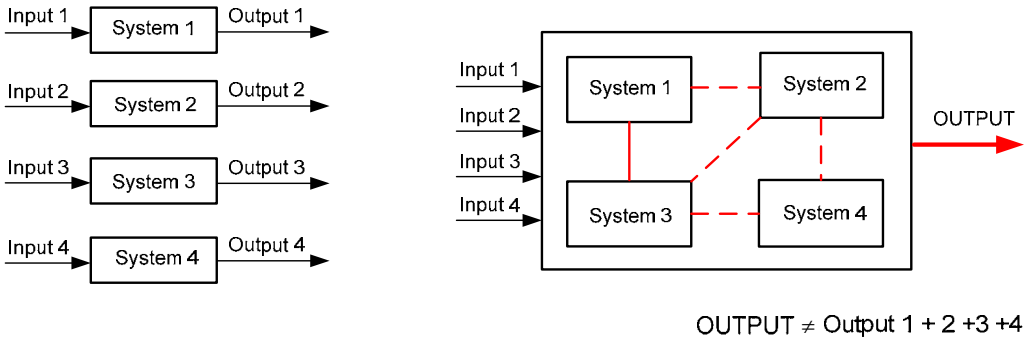


Fig. 2-11. Emergence of automotive System of Systems.

The input and output of a system is predictable. This is because the connection within a system is fixed and very strong so that the causality is determined. But in the SoS, due to the belonging and connectivity characteristics, the structure and interaction inside the SoS are changeable. Therefore, the behaviour of the SoS can not always be predicated by aggregating the inputs and outputs of all individual systems. For example, consider a SoS consisting of four systems which is shown on the left of Fig. 2-11. The inputs and outputs of each system are represented as the arrowed lines. When these systems are grouped together, their output as a

whole SoS does not equal the aggregation of the individual outputs. It is critical to identify and study the interaction and connection of the systems therein in order to predict the behaviour of this SoS. It is especially crucial when testing and proving the behaviour of the SoS. In the case of an automobile, the introduction of a large number of electronic components leads to the emergence of new properties of the automotive electronic SoS which influence the behaviour of such an SoS. The interaction and integration of the automotive electronic SoS have to evolve to accommodate the increasing complexity and other emergent properties.

2.2.3 Summary

Five different characteristics of SoS have been discussed. They are autonomy, belonging, connectivity, diversity and emergence. A large system can be viewed as a SoS when one or more characteristics are satisfied. Such a SoS may include new, modified, or unmodified systems; and some systems may be evolving and their future is unpredictable. These complexities of a SoS cause difficulties in communicating requirements and integration [36]. To handle the development and modification of a SoS consideration needs to be given to the SE and SoSE which are discussed in the next section.

2.3 System Engineering and System of Systems Engineering

SE is an interdisciplinary approach and means to enable the realization of successful systems [37]. The discipline of SE has been recognized for 50 years as essential to the development of complex systems [38]. Since its recognition in 1950s [39], SE has been applied to products as varied as ships, computers and software, aircrafts, environmental control, urban infrastructure and automobiles [40-42]. The need for SE emerged with the increase in complexity of systems and

projects. A system can become more complex due to an increase in size as well as with an increase in the amount of data, variables, or the number of fields that are involved in the design [43, 44]. The development of the automobile is such an example of SE.

SE encourages the use of tools and methods to better comprehend and manage complexity in systems. Some examples of these tools are listed below [45-48]:

- Modelling and Simulation,
- Optimization, System dynamics,
- Systems analysis,
- Statistical analysis,
- Reliability analysis,
- Decision making.

[18] indicates “three evils” of SE: complexity, a lack of understanding and communication issues. Models play important and diverse roles in SE to address “three evils”. A model can be defined in several ways, including [49]:

- An abstraction of reality designed to answer specific questions about the real world.
- An imitation, analogue, or representation of a real world process or structure.
- A conceptual, mathematical, or physical tool to assist a decision maker.

Building the model in the above ways can allow engineers to identify complexity, aid understanding and improve communication. In addition, model-based design integrates modelling into a design, development and validation process as shown in Fig. 2-12. It can be applied to a number of different tools and methodologies [13]. The V-model has been a very popular process in SE and it has

been very successful at playing the role of designing, developing, and deploying new equipment or systems to satisfy specific needs or requirements.

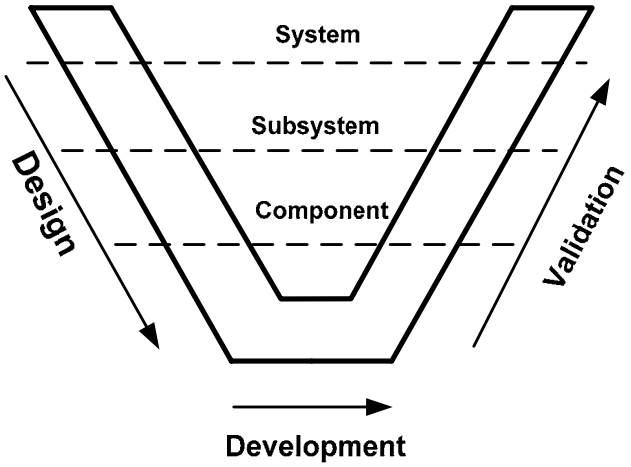


Fig. 2-12. V-model in System Engineering.

At present, systems engineers are facing challenges with the scope, scale, and shape of systems problems in large scale, complex, networked environments. Engineers are increasingly required to expand the capabilities of the system through the integration of systems into SoS to meet various requirements. Because SoS engineers are starting with existing systems with independent owners, objectives and development processes, they are faced with a new set of conditions for their engineering processes [50].

- Increased chance of latent error, bugs, or mismatches.
- Increased number of ways the SoS can fail.
- Decreased user ability to discern failures.
- Increased need for complex systems.

In addition, there are issues beyond complexity that need to be addressed. These include: ambiguity; human social dynamics; sustainability; and methodology [51]. Although SoSE is a term that has been used to represent a set of developing processes and methods for designing and implementing solutions to SoS problems

[52], it has not received universally accepted definition, underlying perspectives related to philosophy, methodology, or standards [53]. [54] has concluded that the current state of SoSE development appears to be bifurcated into two separate paths. The first path engages SoSE from a technically dominated perspective, e.g. interoperability, information technology, net-centricity and technical integration [55]. Producing an "integrated product" is the fundamental purpose. This path utilizes "hard systems" thinking and development and emphasis is placed on objectivity in results and their interpretation. In contrast, the second path is more closely related to "soft systems" thinking, dominated by concerns with human, social, contextual and higher level inquiry to produce purposeful responses to complex system problems [56, 57]. Attention is focused on the interpretative nature of understanding complexities in complex problem domains.

Although there is not a broadly accepted approach, it appears that the convergence can be found in the following points which are representing the primary focus in SoSE [54].

- SoSE involves the integration of multiple, potentially previously independent, systems into a higher level system.
- SoSE enables the collaboration of systems in a SoS and generates capabilities beyond what any of the constituent systems is independently capable of producing.
- SoSE brings systems together in order to perform a higher level mission/purpose for which each member system plays an integral role, but none of the contributing systems can accomplish independently.

It is discussed and indicated in [58, 59] that the SE processes as documented in the SE standards: IEEE 1220, EIA/IS-632, EIA-632, ISO 15288 [60-65], and the

guide: ISO TR 19760 [66], are a necessary and sufficient set of processes to address above objectives in SoSE. However, it has to be recognized that SoSE is carried out under some level of uncertainty and it involves factors in multiple levels and domains. Compared with traditional SE that seeks to optimize an individual system, SoSE seeks to optimize a network of various systems brought together to meet specific requirements [67, 68]. Focusing on the automotive industry, attention has to be given to the new techniques in model-based design in order to allow it to evolve and be capable of managing automotive electronic SoS development which is investigated in this Thesis.

2.4 Automotive system development

This section aims to provide an overview of the automotive electronic system development process. The modelling languages and tools which are used in the development are also discussed in this section.

2.4.1 Overview

Modern cars are now equipped with more and more functionality dependent on embedded electronics, ranging from powertrain and chassis control to body comfort and infotainment. The size and complexity of software for these embedded electronic systems are increasing rapidly with their cost raising from 10% of the overall cost in 1970 to 40% in 2010. 90% of innovations in the automotive industry are driven by electronics and 80% among them are software [69]. Quality is a big challenge in automotive software development. The software failure of automotive systems is severe. Software errors led directly to car recalls. According to the report [70], one-third of the recalls in recent years caused by software errors. More efforts are needed on software verification and testing. Another challenge concerns

the reduction of the development time [71]. The automotive market is shared by manufactures, suppliers and tool vendors, and all need shorted processes which favour the exchangeability among them and reuse of software in different product lines. They also need to follow requirements along the development, from the specification to design and coding, to anticipate and communicate changes throughout teams [72].

2.4.2 Model-based design

Model-based design is a widely used and accepted approach for automotive system development which has been demonstrated in the literature [13, 72-75] and references therein. Model-based design helps address the challenges of embedded system development [16]. Using models at the core of the development process provides engineers with insight into the dynamics and algorithmic aspects of the system through simulation. In addition, the models are also commonly used [76]:

- as executable specifications;
- to communicate (sub-)system requirements and interface definitions;
- to provide virtual prototypes or models of the complete system;
- for automatic code generation of embedded software algorithm or logic.

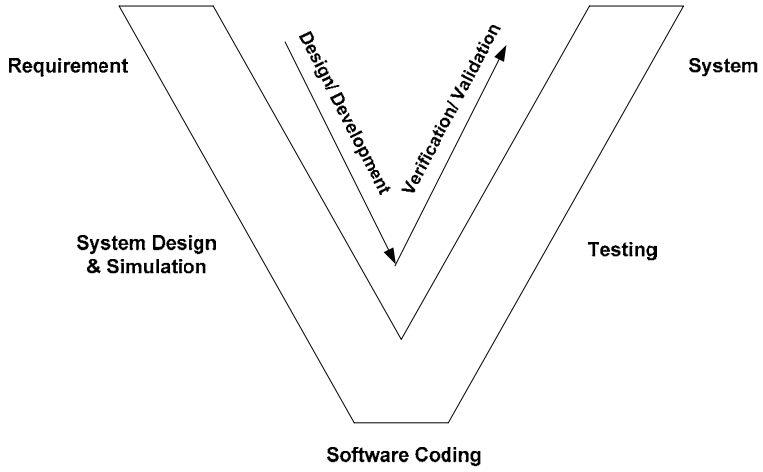


Fig. 2-13. Automotive electronic system development process.

Fig. 2-13 shows the core process of the automotive electronic system development, i.e. a V-diagram. The initial step is the requirement capture. The importance of requirement engineering has been well acknowledged in [77-83]. All projects are driven by requirements. That is, if the requirements are not clearly understood, the system cannot be validated correctly. The modelling language is used to produce the model of the system. Code generated from the model is followed by the system design and simulation. Auto coding is one of the benefits from the model building as it significantly reduces the development cycle. When the code generation is completed, the software and system development and integration can carry on. The development process will then move to the test. In the test stage, the development cycle has to move back to the beginning to check the requirement if there is any error found. This process will be very costly. Consequently, requirement capture, system design and simulation are essential in the whole development process [74]. However, the complexity of automotive systems is increasing. In systems terms, automotive, embedded electronic systems can now be classed as a SoS. The design, implementation and management of such complex distributed systems, and their integration into one cohesive and reliable SoS are presenting new challenges for the automotive industry.

To meet these challenges, it is necessary to investigate and refine the model-based design for capturing the requirements for the SoS, at the outset of the product development process, and conveying the requirements through the stages in the product development process.

As a potential methodology to address the challenges in automotive systems and software development, model-based design has been playing an important role to identify complexity, aid understanding and improve communication [18]. In

order to build the model effectively, it is essential to have a common language to carry out the modelling.

MATLAB/Simulink is a well known modelling and simulation environment. MATLAB is used in a wide range of applications, including signal and image processing, communications, control design, test and measurement. Simulink which is integrated with MATLAB provides an environment for modelling, simulating and analyzing multidomain dynamic systems. In particular, they can be used for model-based design for control systems. Stateflow, the other product developed by the MathWorks extends Simulink with a design environment for developing event-driven systems that contain control and supervisory logic. MATLAB/Simulink/Stateflow support the development and definition of system and software components, their connections and interfaces by graphical models using editable, hierarchical block diagrams and Stateflow diagrams and provide the necessary means of description, computation techniques and interpreters/compilers [84]. Such models can be simulated, i.e. executed. Coupled with Real-Time Workshop (RTW), source code is automatically generated for real-time implementation of embedded systems.

2.4.3 Auto coding and code verification

Traditional automotive software development involves paper designs and hand coding followed by verification activities such as code inspections, structural code coverage analysis, and unit/integration tests. Many of these activities lack tool automation and involve manual interaction. Thus they are error prone and time consuming [85]. Auto coding is a powerful tool for software and system developers. It facilitates the quick and easily source code generation [86, 87]. According to user reports there are increases in efficiency of 20-50% due to model-based

development with automatic code generation in comparison to traditional software development [74].

Furthermore, code checking and analysis tools have recently emerged. They allow software engineers to easily verify the generated code using static analysis techniques. The term static analysis means automatic methods to reason about runtime properties of source code without executing it [88]. It is commonly used during implementation and review to detect software implementation errors. Similar in behaviour to a spell checker or grammar checker in a word processor, static analysis tools detect faults within source code modules. Static analysis has been demonstrated to reduce software defects by a factor of six [89] and detect 60% of post-release failures [90]. Static analysis can detect errors such as buffer overflows and security vulnerabilities [91, 92], memory leaks [93], timing anomalies [94], as well as other common programming mistakes, 40% of which will eventually lead to a field failure. It has been reported that static analysis can remove upwards of 91% of errors within the source code. This analysis can be performed very early in the software design process for finding software reliability breaches before functional tests are performed, or applied later for a sanity check [95, 96].

The first industrial tool for detecting runtime errors using static verification of dynamic properties was PolySpace Verifier [97]. This tool has been commercially available since 1999. It addresses two essential needs of embedded software development:

- Static verification: it statically predicts specific classes of runtime errors and sources of non-determinism.

- Semantic browsing: it statically computes data and control flow to improve program understanding, ease verification and demonstrate the compliance of the program within industry standards (SIL, DO178-B, MISRA, etc).

Run-time errors detected by PolySpace Verifier include:

- Pointer dereferencing issues (null pointers, out-of-bounds pointers).
- Out-of-bounds array accesses.
- Read access to non-initialized data.
- Access conflicts on shared data.
- Invalid arithmetic operations: division by zero, square root of a negative number and so on.
- Overflow and underflow on integers and floating-point numbers.
- Unreachable code.

Static verification checks each code section and provides a detailed diagnostic for each operation that falls into one of four categories:

- Reliable: the operation under consideration will never fail because of a runtime error.
- Incorrect: the operation under consideration will fail.
- Questionable: the operation under consideration may fail under certain circumstances.
- Unreachable: the operation under consideration cannot be activated.

The literature studied indicates that PolySpace demonstrated a superior detection rate [98-101]. PolySpace also has a graphical interface. The tools set PolySpace Viewer can generate and analyse the report, and navigate in the source code. With a static analysis tool like PolySpace integrated into the model-based

design process, the engineer is much more capable of developing flawless software for the automotive electronic system.

2.4.4 Discussion

This subsection discusses how to promptly respond to the rapid growth of in-vehicle, embedded electronic systems in terms of electronic system development.

Model-based design with auto coding has been very successful at playing the role of designing, developing, and deploying new equipment or systems to satisfy specific needs or requirements in system level development. From a study of the literature, model-based design is also considered as a necessary and sufficient set of processes for SoSE. Focusing on automotive electronic system development, the automotive electronic systems can now be classified as a SoS due to a near exponential growth of in-vehicle, embedded electronic systems. Such an electronic system has complex architecture. The development of such a system requires the integration of electronic components and software, the collaboration between the system engineer and the software engineer. Moreover, during the development process, different developers require different pieces of information, depending on what their roles are in the system [18]. Also, for the purpose of analysing a system, it is important to observe a system from many different aspects or viewpoints. Model-based design by using MATLAB Simulink/Stateflow is more capable of handling functional modelling, physical component modelling at a detailed level but it lacks the structural modelling capability. To develop such complex distributed systems, and their integration into one cohesive and reliable SoS requires new modelling languages, tools and methods to refine model-based design technology to suit the development of an automotive electronic system. Against

this background, SysML which is adapted from UML is proposed in this Thesis for requirement capturing and performing structural and functional modelling for automotive electronic system development.

2.5 Systems Modelling Language

The UML is a modelling language with many graphical design notations and it is mainly used for software development. Object orientation is the key feature of UML. It is used to model not only different types of software system structure and behaviour, but also business processes and data structure [22]. The UML specification is defined and maintained by the non-profit computer industry consortium called OMG. The UML offers a standard way to write a system's blueprints, including conceptual things such as business processes and system functions as well as concrete things such as programming language statements, database schemas, and reusable software components [102].

However, there are some problems and challenges with UML some of which are listed below [103]:

- Use cases are not well integrated with other languages.
- Syntactic and semantic overlap within UML, significantly between the classes and components with internal structures.
- Immature constructs require additional effort to eliminate possible bugs.
- Inadequate support for modelling real-time systems.
- Inadequate support for modelling systems and a SoS.

These problems are being addressed by the ongoing evolution of UML. The current standard of UML is 2.0. However, as UML mainly focuses on software development, it is not well-equipped to model entire systems. The SysML focuses

on SE [18]. It is defined by the OMG as “a general-purpose graphical modelling language for specifying, analyzing, designing and verifying complex systems that may include hardware, software, information, personal, procedures, and facilities”. SysML allows engineers to model system requirements, system behaviour and system structure. Hence, it can link the software and other elements of a system, such as hardware, together. In order to address the challenges of increasing software-based systems in automobiles, research is carried out on system modelling with new modelling languages such as the SysML to determine if they are suitable for particular applications.

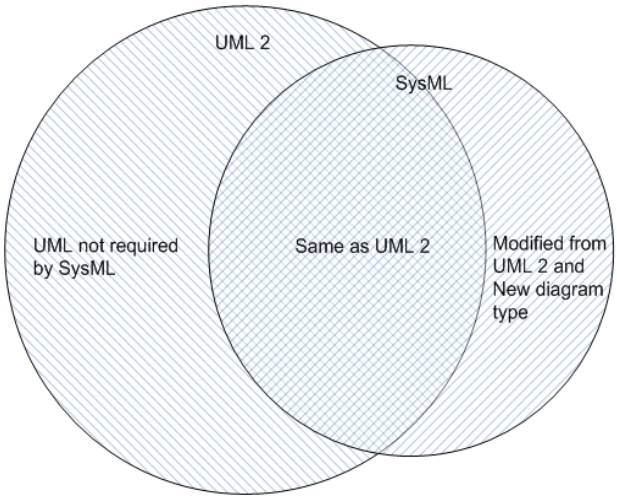


Fig. 2-14. The overlap between the UML and SysML (adapted from [25]).

SysML is based on the UML with additional diagrams and modelling constructs to capture system behaviour and high-level requirements. Fig. 2-14 illustrates the overlap between the UML and SysML. Fig. 2-15 shows the taxonomy of SysML.

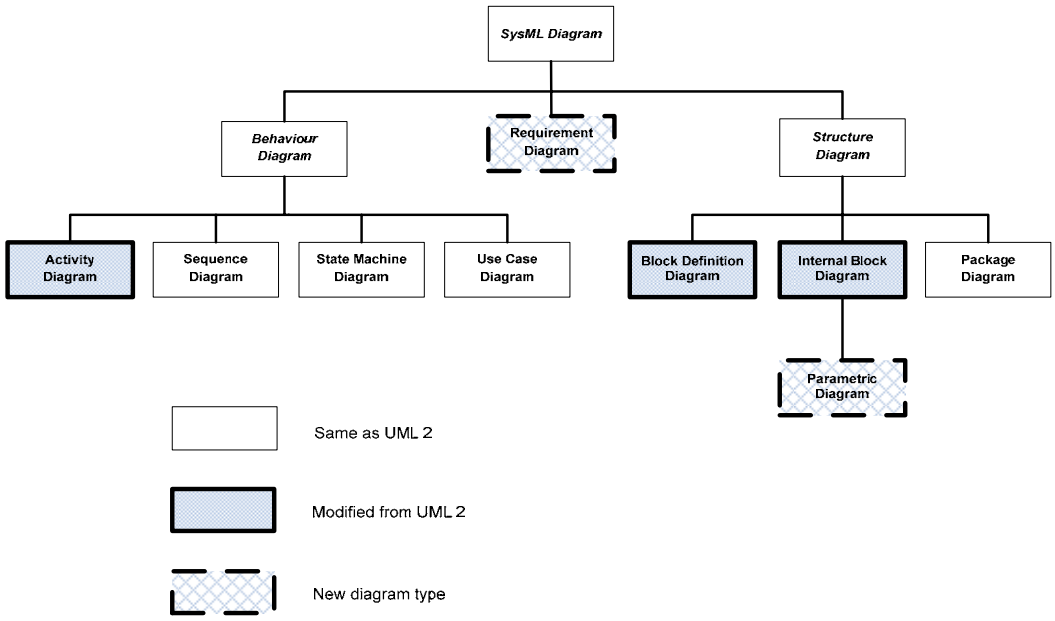


Fig. 2-15. The Systems Modelling Language taxonomy (adapted from [25]).

In SysML, the reuses, extension and addition of UML diagram types are [25]:

- UML diagrams that are reused, but are not extended: use case diagram, sequence diagram, and state machine diagram.
- UML diagrams that are reused and extended: activity diagram (extends UML activity diagram), block definition diagram (extends UML class diagram), internal block diagram (extends UML composite structure diagram), and package diagram (extends UML package diagram).
- New diagram types: parametric constraint diagram and requirements diagram.

The following questions regarding SysML are answered through the research in this thesis.

- Whether SysML, as a modelling language, is well-suited for vehicle manufacturers, and their tier-one suppliers, to exchange information for specifying and clarifying requirements.
- Whether SysML supports software development in a systems engineering

framework.

- Whether SysML adequately supports the development of real-time embedded automotive software.

In summary, SysML is a new modelling language aimed at systems engineers and can in particular be applied to design heterogeneous embedded systems and a SoS. The capability of SysML to model the automotive electronic system and deliver the software is investigated and discussed in the later chapters of this Thesis.

Chapter 3

Driver Information System for the 4×4 vehicle

In Chapter 2, a literature review of Systems and a SoS was presented. The purpose of this chapter is to present the introduction of the vehicle Infotainment System and to provide an overview of the Driver Information System chosen as the pilot study for this Thesis.

3.1 Introduction

The appearance of the automotive Information and Entertainment System has significantly evolved in the past decades. It is driven by the emergence of electronic and networking technologies [5]. In the late 1990s, vehicle information systems like navigation and entertainment systems such as CD players were integrated into vehicles. Today's high-end Information and Entertainment System, one sample of which is shown in Fig. 3-1, can include a variety of features and functions including an integrated 6 disc CD changer, address book, cellular telephone, driver log book, DVD player, navigation system, television, voice

recognition system, or several other similar options [2]. Such systems for information and entertainment are collectively referred to as an “Infotainment System”. The development of such an advanced Infotainment System has benefited from not only networks such as CAN and MOST but also in modern display technology such as Liquid Crystal Displays (LCDs), advanced sensors and the introduction of additional software to manage such a system.

Fig. 3-1. Modern information and entertainment system: 2002 Cadillac CTS.

In order to deliver the best features in new cars, the Infotainment System is now a key focus in the automotive design process. To support the growing importance of the Infotainment System, Land Rover introduced an advanced Driver Information System in its Range Rover 2005 model in June 2004 [104].

As shown in Fig. 3-2, the Driver Information System is centred on a touch screen display located in the mid-fascia area. It forms part of the in-vehicle Infotainment System. A high level display front (HLDF) is the Human Machine Interface (HMI) of this Driver Information System which consists of a 7-inch, 15:9 aspect ratio, 800 × 480 pixel colour display. It has a touch screen membrane over

its display surface. Fig. 3-2 presents a typical display for a vehicle. The detailed description of the Driver Information System is to be provided in a later section.

Fig. 3-2. 4×4 Information System on the vehicle.

3.2 Driver Information System and 4×4 Information System

The following sub-section gives a brief introduction of the features of the Driver Information System and the 4×4 Information System.

3.2.1 Driver Information System

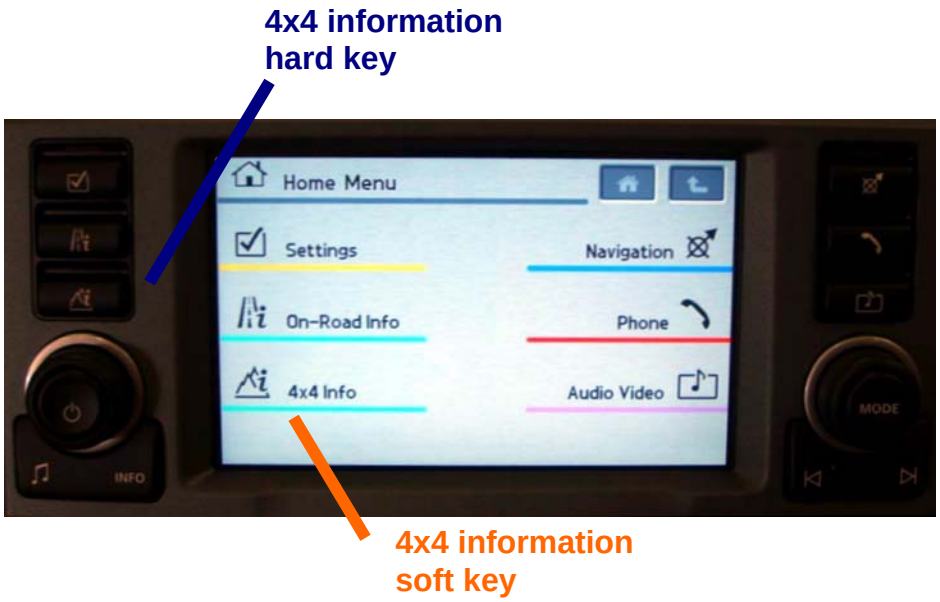


Fig. 3-3. Driver Information System.

Fig. 3-3 presents an actual view of the Driver Information System. By pressing the hard keys and soft keys, the driver can access settings, on-road information, 4×4 information, navigation, phone, audio, video and other features provided by the Driver Information System. Table 3-1 lists the hard keys and their corresponding features.

Table 3-1. Hard keys of the Driver Information System.

Hard Key	Feature	Notes
Home	View home menu	Can also access by soft key
Settings	View settings	Can also access by soft key
On-road	View on-road information	Can also access by soft key
4×4 information	View 4×4 information	Can also access by soft key
Navigation	View navigation	Can also access by soft key
Phone	Access phone	Can also access by soft key
Audio and video	View audio and video information	Can also access by soft key
Mode	Change audio and video mode	

As an example, Fig. 3-4 shows the screen displaying the settings when the driver presses the settings hard key or soft key.

Fig. 3-4. Driver Information System settings.

As a Driver Information System for a 4×4 vehicle, the key function of this system is to provide the 4×4 information such as steering angles, vehicle heights and Terrain Optimization (TO) settings to the driver to enable better off-road driving. The work described in this Thesis concentrates on the 4×4 Information System incorporates in the Driver Information System. The features of the 4×4 Information System are presented in the following section.

3.2.2 4×4 Information System

The features of the 4×4 Information System on the vehicle are summarized in the table below.

Table 3-2. Features of the vehicle.

Number	Feature	Notes
Feature 1.	Display Steering Angle Information	Display front road wheel angle by 13 different images.
Feature 2.	Display High/Low ratio selection status	Display High or Low ratio selection status for transfer gear
Feature 3	Display Gear Position	Display gear position such as “P R N D 5 4 3 2 1”
Feature 4a.	Display Differential lock Information - Centre	Lock or Unlock
Feature 4b.	Display Differential lock Information - Rear	Lock or Unlock
Feature 5.	Display Terrain Optimization Mode	Display one of five different Terrain Optimization settings
Feature 6.	Display Hill Descent Control Status	Display Hill Descent Control Status such as “Inactive, Set, Pending”
Feature 7.	Display Air Suspension Status	Display Air Suspension Status such as “Off-Road, Standard, Access”
Feature 8.	Display Wheel Height Status	Display Wheel Height Status by different images

Feature 1 provides information on how steering angle changes with movements of the steering wheel. Graphics represent the front road wheels' angle in the plain view.

Feature 2 is for displaying the transfer gear status, for example whether a High or Low ratio has been selected. When a range selection is performed, the appropriate graphic will be displayed in the chassis map.

Feature 3 is for displaying a letter or number on the graphics to indicate the gear position. The letters and numbers and what they represent are listed below respectively: "P" for Park, "R" for Reverse, "N" for neutral, "D" for Drive and a number within the 1 to 5 range indicating the gear selection.

Feature 4 is for displaying the differential lock information for both centre and rear differentials. The vehicle has to be fitted with a centre electronic differential (e-diff) as standard and a rear one as an option. The vehicle will then be fully capable of automatically determining through the Traction Control System where to best distribute the torque. E-diff delivers superior on and off-road traction by the cross wheel slip without the need for sudden intervention from the brakes. E-diff can act sooner and more subtly than the traditional traction control methods that rely on the ABS [105].

Feature 5 is for displaying the TO mode selected. The vehicle can provide five different driving modes which are Standard, Grass / Snow / Ice, Mud / Ruts, Sand and Rock Crawl. In one of the four special terrain mode settings listed above, the rear differential locking is actively controlled by the Driveline Control ECU. When one of the special programmes is selected, the pre-load locking torque of the differential is modulated according to the TO setting.

The purpose of Feature 6 is to display the Hill Descent Control (HDC) status. HDC is used to provide a smooth and controlled hill descent in rough terrain.

Feature 7 is for displaying the air suspension status. The air suspension has three suspension heights, i.e., Off-road, Standard and Access.

Feature 8 is for displaying the wheel height status such as whether the vehicle is in standard height. It can also indicate whether the vehicle height is currently changing such as “Rising”.

The key functions of this system include the TO settings, HDC, wheel height display, air suspension heights, steering angle, transfer gear, gear position and differential lock. The detailed functionality of the 4×4 Information System can be found in Appendix A.

3.3 Driver Information System architecture

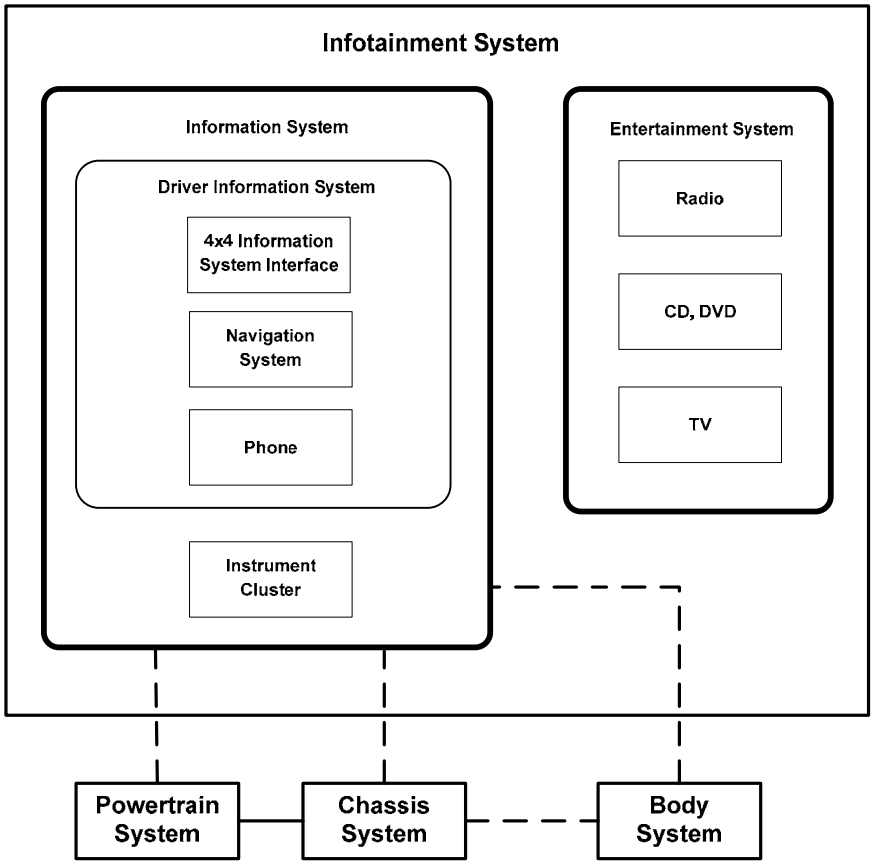


Fig. 3-5. Automotive System of Systems.

Fig. 3-5 presents the architecture of the Driver Information System. The Driver Information System forms part of the in-vehicle Infotainment System and provides the driver and/or front passenger with the ability to view information and status relating to: the instrument cluster; the navigation system; the front and rear entertainment systems; the in-vehicle phone; and the 4×4 Information System. This is achieved by presenting information within the Infotainment System which is broadcast via a MOST data bus, or information from the Powertrain System, Chassis System and Body System which is transmitted on a CAN data bus. The MOST and CAN buses are linked via a gateway located in the instrument cluster.

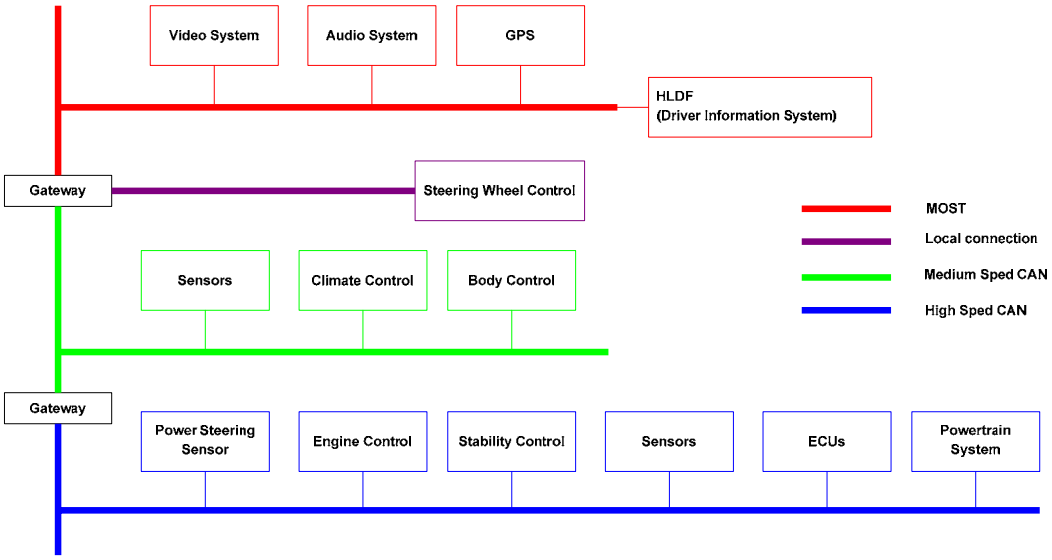


Fig. 3-6. Driver Information System architecture.

Fig. 3-6 presents some key systems and elements of a typical Driver Information System. From this diagram it can be seen that the Driver Information System is hosted within the HLDF on the MOST network. All the information displayed in the Driver Information System is obtained from several relevant on-board vehicle electronic systems such as various sensors and ECUs in different networks.

Table 3-3. Driver information on networks.

Network	**Driver Information Obtained**
MOST BUS	Audio and Video relating to navigation, radio, etc.
CAN BUS	Vehicle Height, Gear Position, etc.

Table 3-3 lists an example of the driver information which is obtained from different networks on the vehicle. From Fig. 3-6, it can be also seen that the information from the Powertrain System is delivered on a high speed CAN. This vehicle contains two separate CANs operating at different transmission rates. The medium speed CAN running at less than 125 Kbps usually manages a vehicle's

"comfort electronics," like seat and window movement controls and other user interfaces. Generally, control applications that are not real-time critical use this medium-speed network segment. The high speed CAN runs more real-time critical functions such as engine management, ABS, and cruise control [106]. Focusing on this 4×4 Information System, gear position, differential lock information and other information from the chassis and powertrain that are vehicle status related are obtained from the high speed CAN. MOST facilitates the deployment of a digital data infrastructure in the vehicle for these advanced audio/video systems.

3.4 Characteristics of the 4×4 Information System

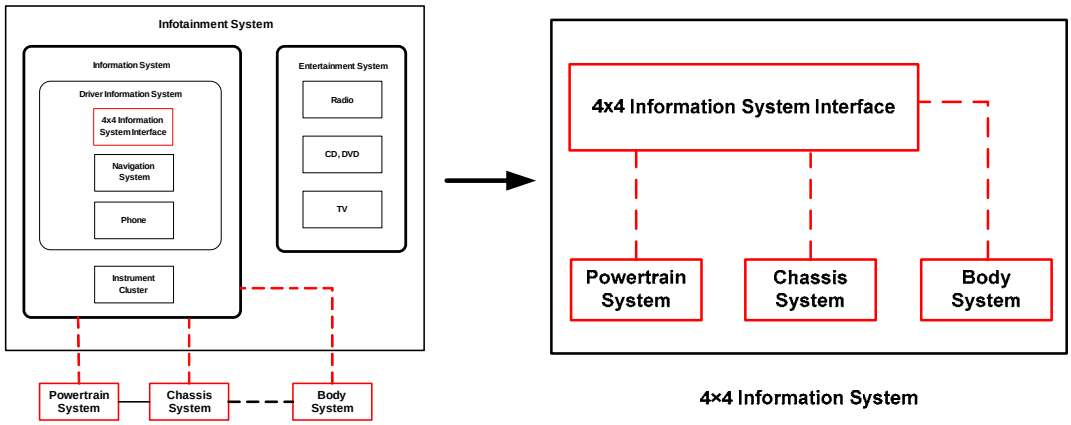


Fig. 3-7. The 4×4 Information System.

Fig. 3-7 represents the 4×4 Information System within the automotive SoS. The 4×4 Information System, accessed through the colour touch-screen on the fascia, provides real-time graphical indications of essential 4×4 information to the driver to deliver the best possible off-road driving in difficult conditions. The HLDF shows the direction of the front wheels, air suspension height settings, activation of the HDC, the high/low gear ratio selection, wheel articulation, the compass view and the TO mode. As shown in Fig. 3-6, this system has to obtain information from

several other on-board vehicle electronic systems such as various sensors and ECUs. The ECUs are linked by a communications network, consisting of several data bus technologies with exponentially increasing transmission rates.

After studying the physical structure and functionality of the 4×4 Information System, the following characteristics of SoS can be identified from this 4×4 Information System:

Autonomy

As shown in Fig. 3-7, the Powertrain System, Chassis System and Body System have independent functions. For example, the Chassis System takes a driver's TO settings and changes the air suspension to the corresponding height. It has the ability to maintain its function independently.

Belonging

The 4×4 Information System Interface includes the physical interface and application software. It chooses the Powertrain System, Chassis System and Body System to form this SoS. The data are captured from various sensors and ECUs in these systems which are chosen to belong to this SoS. Thus, the 4×4 Information is displayed to the driver.

Connectivity

As described in Chapter 2, some connections in the SoS can be added, removed or changed. In the 4×4 Information System, enabling additional connections between the 4×4 Information System Interface and the Powertrain System, Chassis System and Body System allows more data and information to be delivered. For example, enabling the connection between the steering angle sensor and the 4×4 Information System Interface allows the steering angle of the vehicle to be displayed in the 4×4 Information System.

Diversity

The driver's various requirements derive the diversity of the 4×4 Information System. The 4×4 Information System does not only display the necessary 4×4 information such as gear selection, suspension height but also displays compass information and vehicle settings. Data captured on different vehicle networks at different speeds and amounts have to be managed and displayed to the driver within the 4×4 Information System.

Emergence

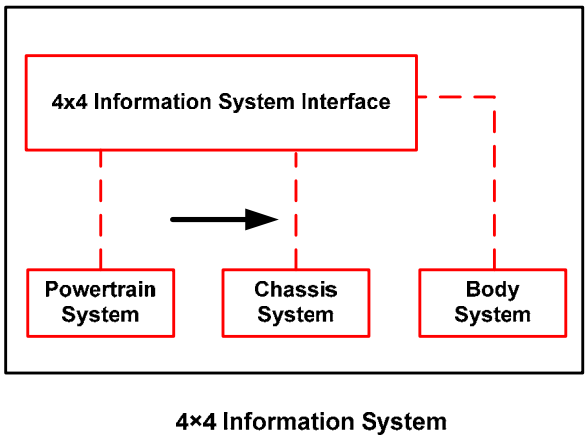


Fig. 3-8. The 4×4 Information System.

From the description of physical structure and functionally of the 4×4 Information System, it can be taught to enable a single function in the 4×4 Information System that involves data capture from the sensors of the Powertrain System, Chassis System or Body System and data transfer on several data buses. As shown in Fig. 3-8, the arrowed line indicates the connection between the steering angle sensor in the Chassis System and the 4×4 Information System Interface. Displaying the steering angle in the 4×4 Information System contains the data capture from the steering angle sensor and data transfer to local connection and then data delivery on the gateway and MOST bus. Both physical structures of

the vehicle network and software function of the 4×4 Information System have to be considered and developed in order to correctly display the large mount real-time data and enable the advanced functions of the 4×4 Information System.

Therefore, having identified the above characteristics, the 4×4 Information System is concluded as a SoS. The development of this SoS through a model-based design approach is to be presented in the next chapter.

3.5 Discussion

This chapter has described the physical structure and functionally of the Driver Information System and the 4×4 Information System. The characteristics of the 4×4 Information System are identified and discussed that show the 4×4 Information System which is part of the in-vehicle Infotainment System is a typical SoS. Similarly, the Chassis System and Powertrain System utilize the networked electronic systems to form the automotive electronic SoS to achieve advanced electronic control and others functions. Therefore, the experience gained from the development of this SoS by utilizing new techniques can benefit the development of other automotive SoSs. In addition, these techniques are generic and they are applicable in SE and SoSE. In other words, they can be adopted for the development of aerospace systems and the hybrid electric vehicle (HEV). However, the 4×4 Information System is not a safety critical system for the vehicle. As a consequence, further investigation is required for applying the experience gained from this study to a safety critical system such as adaptive cruise control on the vehicle.

Chapter 4

Modelling of the 4×4 Information System using SysML and MATLAB Simulink/Stateflow

In chapter 3, an overview of the Driver Information System and its 4×4 Information System was presented. The purpose of this chapter is to provide a detailed model of the 4×4 Information System chosen as the pilot study for this Thesis.

4.1 Introduction

After the functional description of the 4×4 Information System has been described in relation to its software and hardware elements within the overall automotive electronic system in Chapter 3, this chapter describes research into model-based development of the 4×4 Information System. Two distinct model-based approaches to automotive electronic system development are explored. The first approach involves the use of the SysML based tool ARTiSAN Studio for structural

modelling and functional modelling. The second approach involves the use of the MATLAB based tools Simulink and Stateflow for functional modelling. Both approaches for developing the model of the 4×4 Information System are critically evaluated. The strengths and weaknesses of the different approaches are explored and compared. Conclusions are drawn about how the model can benefit the development of such an automotive electronic system.

4.2 Model built in SysML

This section will discuss how SysML is used to provide an architectural description of the 4×4 Information System. Specifically, this section will explore: the use of block definition and internal block diagrams, for structural modelling of the system; and the use case, sequence, state machine and activity diagrams for modelling the functional behaviour of the system. Finally, this section will summarize how to make use of SysML for the development of an automotive electronic system.

4.2.1 The modelling process

The software field is developing rapidly. New areas of practice and research are emerging with an ever increasing speed [107]. It is believed to be beneficial for a project to introduce to the developers the concept of software architecture [108-112]. [113] has concluded that the viewpoints and views are well-established concepts in software architecture. For this reason, the IEEE Standard 1471-2000, “Recommended practice for architectural description of software intensive system” is introduced in this Thesis for guiding the construction of the SysML model. View and viewpoint are central concepts in the IEEE 1471 Standard for architectural description [114]. According to IEEE 1471, a view is a representation of a whole system from the perspective of a related set of concerns. Concerns are those

interests, which pertain to the system's development, its operation or any other aspects that are critical or otherwise important to one or more stakeholders. A viewpoint is a pattern or template from which to develop individual views. It establishes the purpose and audience for a view and the techniques for its creation and analysis. In order to satisfy the IEEE 1471 standard a viewpoint should specify at least:

- A viewpoint name.
- The stakeholders the viewpoint is aimed at.
- The concerns the viewpoint addresses.
- The language, modelling techniques, or analytical methods to be used in constructing a view based upon the viewpoint.

Two viewpoints are selected in this Thesis, namely, a structural viewpoint and a behavioural viewpoint. They are specified in the Table 4-1 and Table 4-2.

Table 4-1. Viewpoint specification--Structural viewpoint.

Viewpoint name	Structural viewpoint
Stakeholders	System developer
Concerns	What elements compose the system? How do they interconnect? What are the mechanisms for interconnection?
The modelling language to be used	SysML
View(s) to conform this viewpoint	Physical structure view

Table 4-2. Viewpoint specification--Behavioural viewpoint.

Viewpoint name	Behavioural viewpoint
Stakeholders	System developer
Concerns	What are the dynamic actions of and within a system? What are the kinds of actions the system produces and participates in? How do those actions relate (ordering, synchronization, etc.)? What are the behaviours of system components? How do they interact?
The modelling language to be used	SysML
View(s) to conform this viewpoint	Function view

As shown in Table 4-1 and Table 4-2, two viewpoints are selected based on the consideration of the stakeholders to whom the architectural description is addressed and their concerns. Specifically, the Thesis chooses the system developer as the stakeholder. Two viewpoints are aimed at the system developer's concerns, e.g. "What elements compose the system?", "How do they interconnect?" and "What are the behaviours of system components?" SysML is chosen as the modelling language to provide an architectural description of the 4×4 Information System from a structural viewpoint and a behavioural viewpoint. Physical structure view and function view are constructed to conform to the structural viewpoint and the behavioural viewpoint respectively. Each view consists of one model which is described in detail in the next section.

4.2.2 Structure model

Structural modelling is performed to provide a physical structure view of the 4×4 Information System. Block definition diagrams and internal block diagrams are created to depict the physical structure of the 4×4 Information System.

4.2.2.1 Block definition diagram

Block definition diagrams define a class of objects with similar properties, behaviour and interactions. During the practical model development process, it is conventional to start thinking about objects required in the model at the beginning. When a set of objects is found to have the same properties, behaviour and interactions, a class should be defined for these objects.

In the 4×4 Information System, various signals and information have to be delivered and transferred through several networks. These networks have different architectural levels in the system. They also have different properties to support data transfer in the system. The role they play in the 4×4 Information System has to be clearly described by defining their attributes and operations in the system.

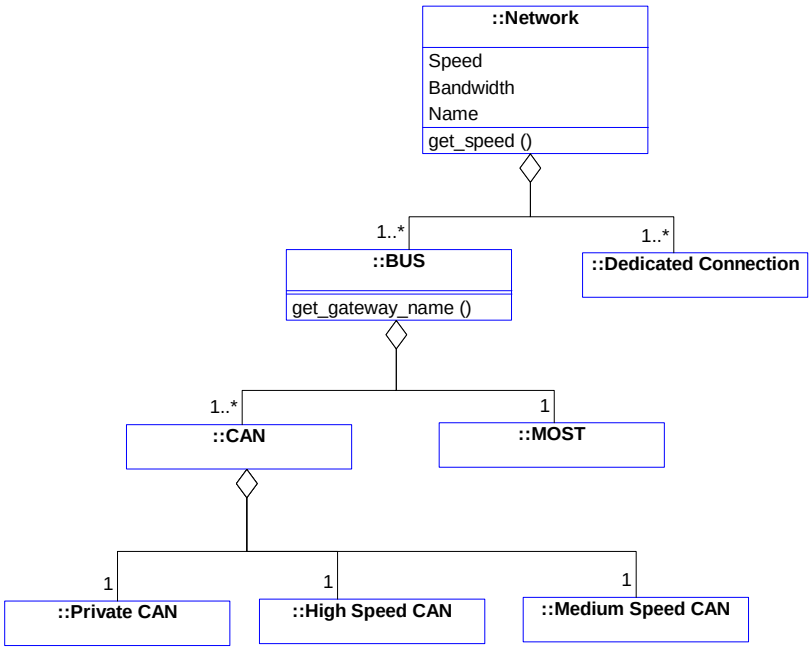


Fig. 4-1. Network class.

In Fig. 4-1, the network class of the 4×4 Information System is defined. This network includes three types of CAN bus which are the private CAN bus, the high speed CAN bus and the medium speed CAN bus. CAN buses and the MOST bus form the bus class. The attribute of this bus class includes “name”, “speed” and “bandwidth”. Two operations are indicated which are “get_speed” and “get_gateway_name”. Different buses and the dedicated connection are integrated to provide several transmission rates for data transfer in the 4×4 Information System. The network class represents the highest level within the structure model. The model also includes four further classes representing lower levels within the network architecture.

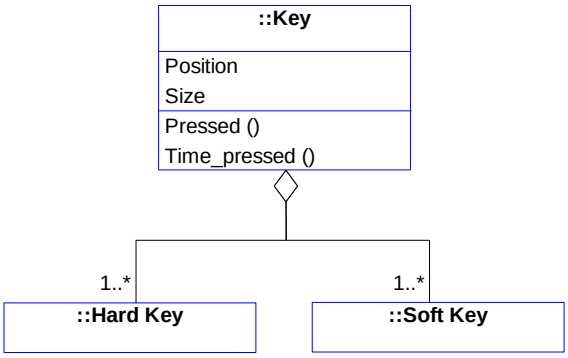


Fig. 4-2. Key class.

For example, Fig. 4-2 shows a block definition diagram representing the key class that models the operations of keys which are used to select functions within the 4×4 Information System. Two different types of keys are included and they are a “Hard Key” and “Soft Key” respectively. They are shown as subclasses of the “Key” class. The “Key” class has the attributes of “position” and “size” which are used to describe the appearance of the key. The operations of the key class use “Pressed” and “Time_Pressed” to identify whether the key has been pressed and how long it has been pressed.

4.2.2.2 Internal block diagram

Internal block diagrams are useful for modelling the decomposition of structured classes through parts, and showing the connections between those parts. Compared to the block definition diagram, the internal block diagram focuses on the detailed level of the system structure. Two internal block diagrams have been developed to present interactions at different levels of abstraction among the 4×4 Information System and other systems of the vehicle.

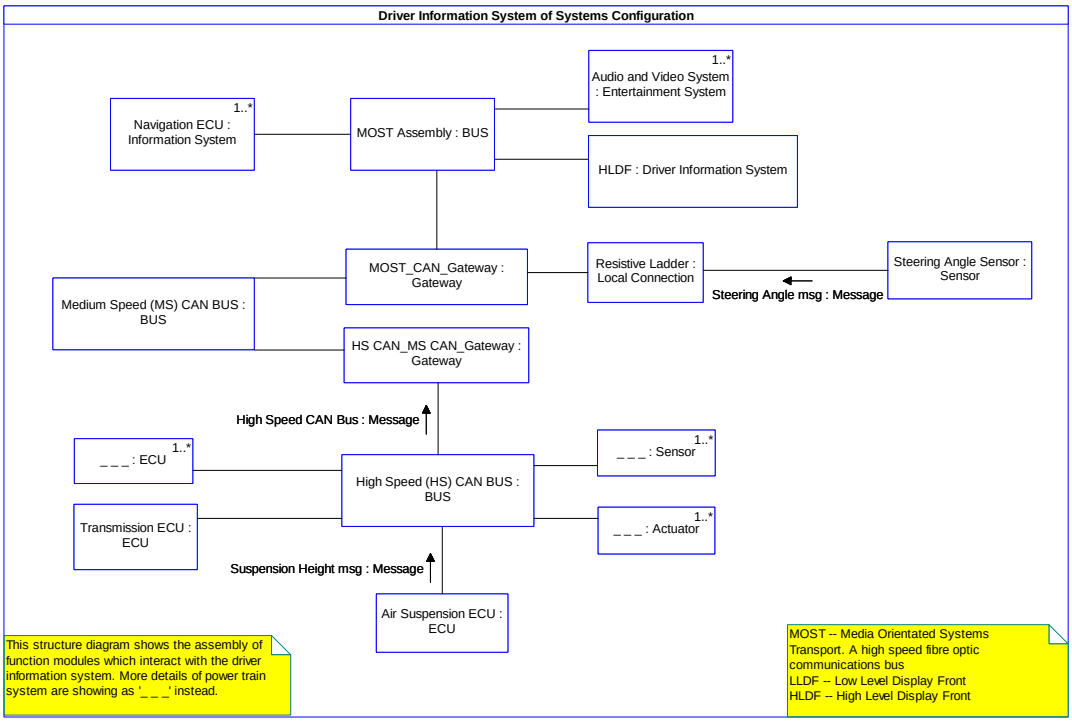


Fig. 4-3. Internal block diagram: Driver Information System overview.

The first internal block diagram, shown in Fig. 4-3, is utilized to provide a high level description of the Driver Information System. It includes a description of the MOST and CAN buses and the gateways that enable communication between the buses. The location of the Driver Information System is clearly displayed. In this diagram, each rectangle represents a part which can be either a function module or a hardware module. The text to the right of ":" in the rectangle is the type name of the part. The part name is specified to the left of ":" in the rectangle. When a part

has a multiplicity more than one, it is shown in the top right of the part. For example, there are numerous sensors, ECUs and actuators on the high speed CAN bus so their numbers are indicated as “*” in the rectangle respectively. Gateways which are used to provide the communication between data buses are also represented in this diagram. Lines connect different parts together to show the interactions among them. Arrows are used to define the direction of the data transfer.

The air suspension ECU sends the suspension height information to the high speed CAN bus in the direction indicated by the arrow. Afterwards, it is delivered to the medium speed CAN bus through the “HS CAN_MS CAN_Gateway” together with the signals from other ECUs and various sensors and actuators on the high speed CAN bus. For example, centre and rear differential lock data and gear selection data are captured by the transmission ECU on the high speed CAN bus. The data which show the level of cross-axle articulation, the status of the HDC and the TO settings is also collected by the ECUs on the high speed CAN bus. The information is transmitted to the MOST bus through the “MOST_CAN_Gateway”. Additionally, the steering angle data are captured from the steering angle sensor and directly delivered to the “MOST_CAN_Gateway” by the local connection, and transmitted to the MOST bus. Other information such as data used in the compass view is transmitted from the navigation ECU over the MOST bus that supports the Infotainment System. The 4×4 Information System receives the data from the MOST bus and displays it on the screen according to driver’s selection.

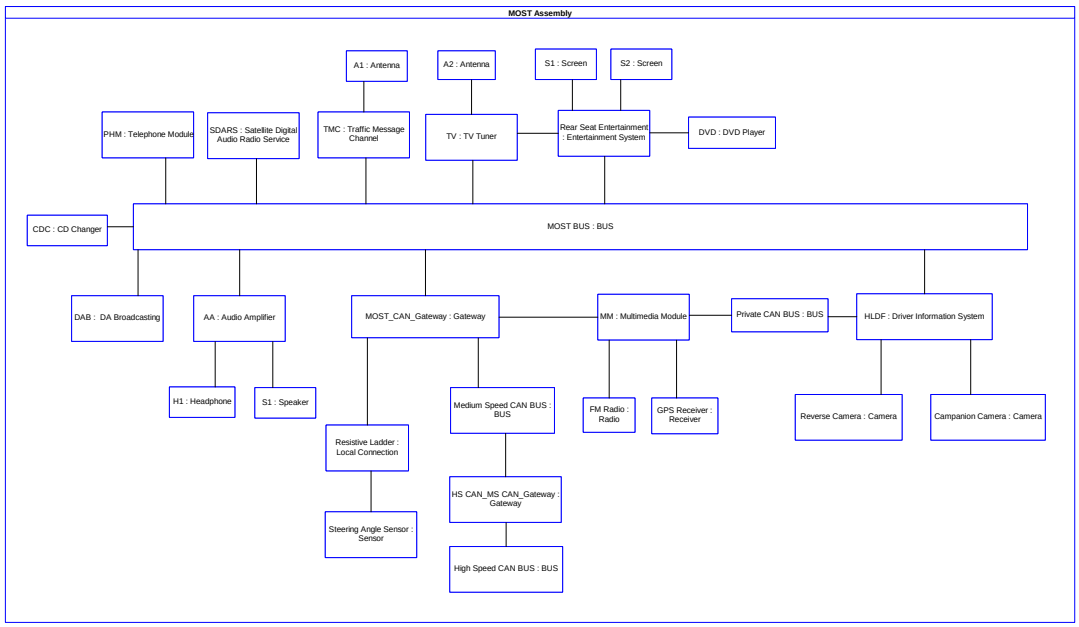


Fig. 4-4. MOST System of Systems overview.

The second internal block diagram shown in Fig. 4-4 is a detailed level description of the vehicle network configuration. It represents the assembly of data buses, gateways, systems and hardware modules which are connected or related to the Driver Information System. In this diagram, each rectangle represents a part which can be either a function module or a hardware module. Lines connect different parts together to show the interactions between them.

For example, connections between different systems on the MOST bus and other gateways to buses like the medium and high speed CAN are clearly represented in this diagram. Specifically, the steering angle data are delivered to the MOST_CAN gateway through the local connection. Then the MOST_CAN gateway sends the data to the MOST bus and they will be taken and displayed to the driver by the 4×4 Information System when associated keys are pressed. Besides, radio, telephone, navigation and other components of the Information System deliver data to the MOST bus. Meanwhile, the Entertainment System transmits audio and video signals to the MOST bus such as a video signal from the

DVD player. Lines between cameras and the Driver Information System show the data from cameras is delivered to the Driver Information System directly. The type and amount of the data transmitted on the network are presented in this diagram. Moreover, according to the internal block diagram shown in Fig. 4-4, the bandwidth, speed and other attributes of the communications network can be considered and defined within classes of the block definition diagrams. The type of the data transmitted, its bandwidth and other attributes are defined within those block definition diagrams.

4.2.3 Function model

A function model is developed to represent the functional behaviour and provide a function view of the 4×4 Information System. The use case, sequence, state machine and activity diagrams are utilized in the function model.

4.2.3.1 Use case diagram

Use case diagrams are used to specify the functionality in terms of high-level requirements. They depict a system's behaviour in terms of its responses to requests that come from outside, for example, from the driver. There are three elements in a use case diagram: use case, actor and relationship, respectively. Each requirement is represented as one use case in the diagram and actors can be either people or a system. A use case can be defined at several levels and there are three use case diagrams in the model.

Fig. 4-5 shows the highest level use case for the 4×4 Information System. It shows the interaction between the front occupant including the driver and front passenger and the different components of the system. “Access 4×4 information” is the top level use case in this diagram and it includes many lower level use cases. Through the top level use case, the driver can have access to “View steering”,

"View gear position", "View Hi/Lo ratio", "View suspension information", "View Diff Lock", "View Compass", "View Home", "View Terrain Optimizations Settings" and "View Hill Descent Control Status" use cases directly. Arrows with an 'include' label indicate the functional structure of the 4×4 Information System. In this way, the functionality can be broken down to different levels. In this use case diagram, the driver can go to "View Differential Lock Rear" and "View Differential Lock Centre" from the "View Diff Lock" use case. The driver can also go to "View whether Standard/Sand/Rock Crawl/Mud" which is the lower level use case of the "View Terrain Optimizations Settings" use case.

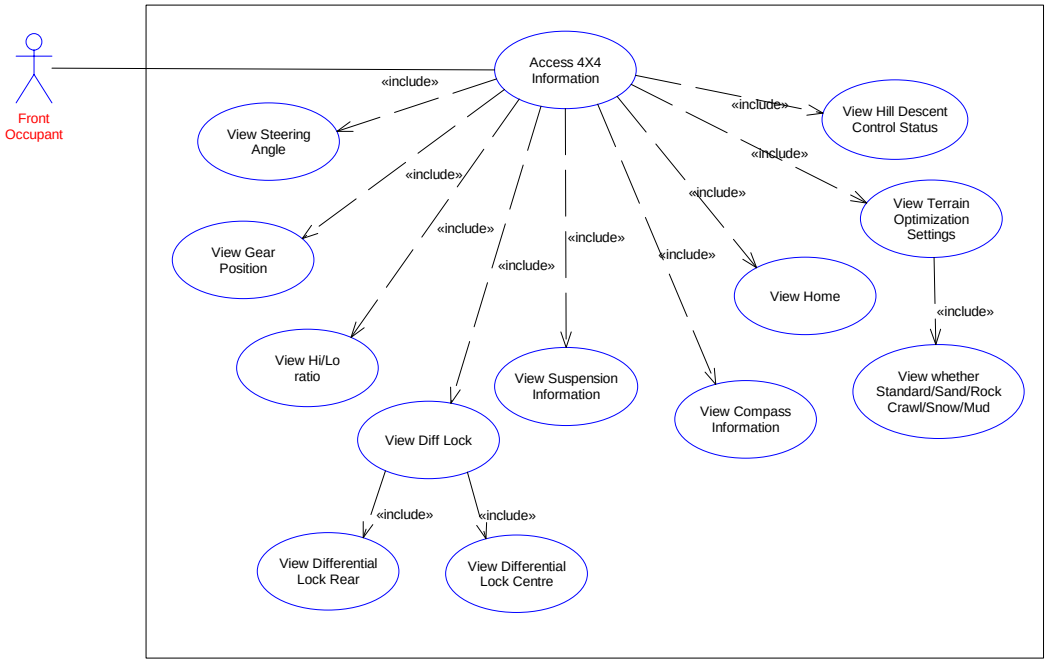


Fig. 4-5. Use case diagram: 4×4 information use case.

In order to make the diagram clear, it is not recommended to put too many details in one use case diagram. If one use case has many sub functions that need to be broken down to a detailed level, another use case diagram can be developed. Within this model, a separate use case diagram shown in Fig. 4-6 is developed to specify the "View Suspension Information" use case in Fig. 4-5. In this diagram, the "View Suspension Information" use case includes three lower level use cases

which are “View vehicle height”, “View whether vehicle raising/lowering” and “View chassis height”.

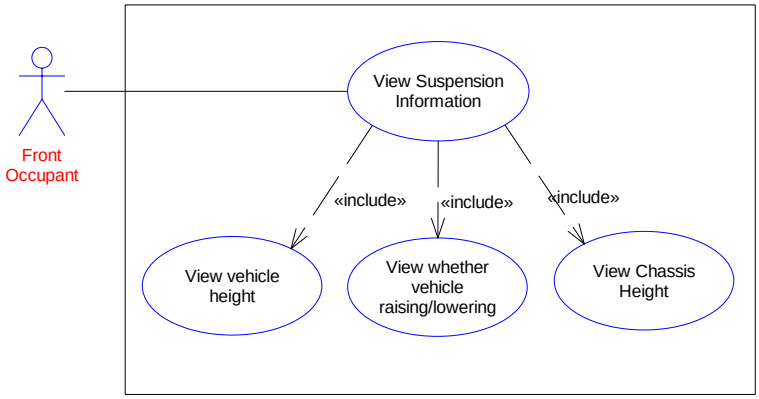


Fig. 4-6. Suspension information use case.

Three use case diagrams in the model represents all the use cases in the 4×4 Information System graphically. The use cases presented in the diagrams capture and interpret all the usage scenarios in the specification document of the 4×4 Information System. [13] highlights the importance of employing the textual use case in capturing detailed behavioural requirements of the automotive electronic system. The textual use case is made up of main flow that specifies how the external actor can interact with the system to achieve the desired objectives of the system. Moreover, the alternative flows are defined to represent the other means by which the system may interact with the external actor to accomplish their objective [115]. The conventional requirements capture process mainly concentrates on how the system should behave under ideal conditions, i.e. main flow. Utilizing the use case template introduced in [13] ensures that at each stage of the main flow, the developers are forced to investigate not just what may go wrong with the system, i.e., the error flows, but also how the system could meet the objectives of the system alternatively, i.e., the alternative flows.

Table 4-3. Use case text - Display air suspension status.

Title	Display air suspension status.
Preconditions	The display is showing the home menu screen.
Trigger Action	The driver chooses to view air suspension status.
Objective	4×4 Information System displays the current air suspension status and updates the display when a new air suspension height is selected.
Main Flow	1. The diver presses the 4×4 information hard key or soft key. 2. The 4×4 Information System displays the current air suspension height to the driver.
Alternative Flows	1. The diver presses the 4×4 information hard key or soft key. 2. The diver selects access mode. 3. The 4×4 Information System validate that the vehicle speed is less than 50mph. 4. The air suspension changes to access height and the 4×4 Information System displays the access mode to the driver.
Error Flows	None.
Success Guarantee	The current air suspension status is displayed and updated with new air suspension height selections.
Minimum Guarantee	Displays current air suspension status when a new air suspension change request is not approved.

Table 4-3 shows an example of a textual use case which the use case of display air suspension status is described in detail. This table lists preconditions, trigger action and the objective of this use case. How the system should behave under the different conditions is defined in the main flow and the alternative flow. The error flow is not presented in this table because there is no information available in the usage scenarios in the specification document of the 4×4 Information System regarding what may go wrong with the system. The success guarantee in this table states the assertion of the successful running use case. The minimum guarantee

describes the fewest actions the system should perform, particularly when the primary goal cannot be delivered.

Table 4-4. Use case text - Display compass information.

Title	Display compass information.
Preconditions	The display is showing the home menu screen.
Trigger Action	The driver chooses to view the compass information in the 4×4 Information System.
Objective	The 4×4 Information System displays compass information when compass view is selected in the 4×4 information screen.
Main Flow	1. The diver presses the 4×4 information hard key or soft key. 2. The driver presses the compass view soft key. 3. Compass information is displayed in the compass view on the screen.
Alternative Flows	None.
Error Flows	1. The diver presses the 4×4 information hard key or soft key. 2. The driver presses the compass view soft key. 3. A GPS signal is not available thus the compass information cannot be displayed on the screen.
Success Guarantee	Compass information is displayed in the compass view on the screen.
Minimum Guarantee	Compass view is displayed on the screen without compass view information.

The usage scenarios in the specification document of the 4×4 Information System lacks a description of how the system should behave when the condition goes wrong. Table 4-4 demonstrates how the error flow is investigated during the construction of textual use cases. This table represents the use case of the display compass information. The 4×4 Information System should display compass information following the main flow in this table under ideal conditions. However,

the investigation has to be carried out on how the 4×4 Information System should behave when the GPS signal is not available as described in the error flow.

The attention of the following sections focuses on how the use cases are described in detail within other diagrams of SysML in the function model.

4.2.3.2 Sequence diagram

The development of sequence diagrams is a particular path through a use case. Sequence diagrams are used to represent the interactions which occur among various objects involved in realizing the functionality as per use cases. As a result, one use case may have several sequence diagrams associated with it. There are 13 sequence diagrams in this model at both high and detailed levels.

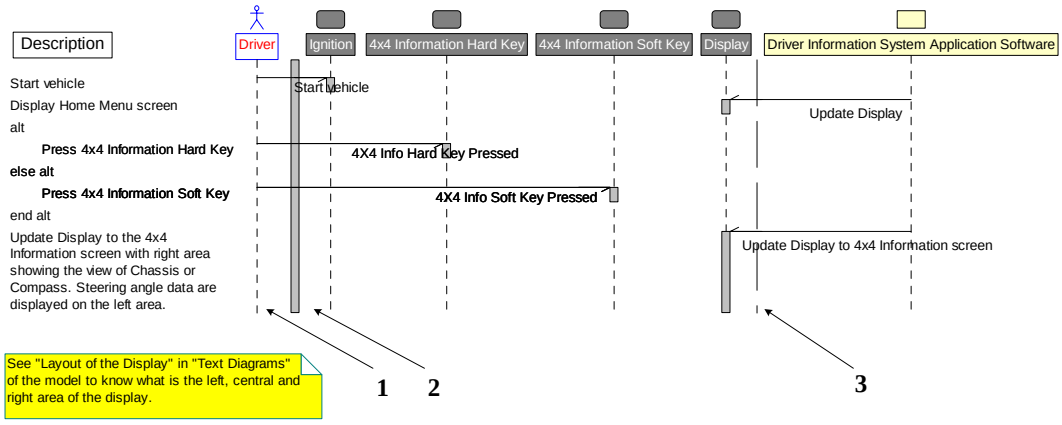


Fig. 4-7. Sequence diagram: view steering angle (high level).

Fig. 4-7 is a high level sequence diagram; it shows the sequence involved in how the driver views the steering angle. In this diagram, objects are defined across the top of the diagram and modelled as vertical lines (marked as “1”). Objects could be actors, class instances, parts, interface devices, packages or subsystems. The description of the sequence is listed on the left side of the diagram. The thick grey vertical line (marked as “2”) separates external actors from elements within the system. To the immediate right are interface devices until the vertical dashed line (marked as “3”). The “Driver Information System Application Software” is at

the right hand side of the vertical dashed line. The vertical time axis is nonlinear and can be regarded as event driven. Arrow lines are used to show how data flow transfer occurs in the system in a time sequence. As shown in Fig. 4-7, the driver starts the vehicle at the beginning. The Driver Information System application software will update the display to the home menu screen. As soon as either the 4×4 information hard key or 4×4 information soft key is pressed, the Driver Information System application software will update the display to the 4×4 information screen. The description on the left of the diagram indicates where the steering angle is displayed.

In order to show how data flow is obtained and transferred between the ECUs, gateways, buses and other physical components, a detailed sequence diagram was developed as seen in Fig. 4-8. It shows how information is captured from the different sensors and ECUs and how the Driver Information System obtains the information from the MOST bus.

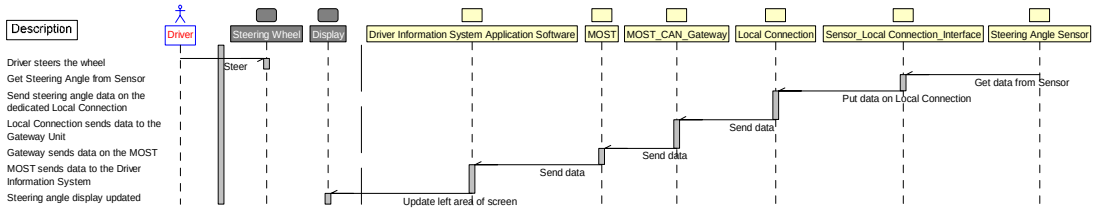


Fig. 4-8. Sequence diagram: view steering angle (detailed level).

Fig. 4-8 shows that when the driver steers the wheel, information is collected from the steering angle sensor and transferred to the local connection through the “Sensor_Local Connection_Interface”. The local connection delivers the data to the “MOST_CAN_Gateway”. Then the MOST_CAN_Gateway sends the data to the “MOST” and it will be received by the Driver Information System application software. Afterwards, it is displayed in the left hand area of screen. The entire route

is shown clearly in the diagram. The action which each object takes is also marked with arrow lines.

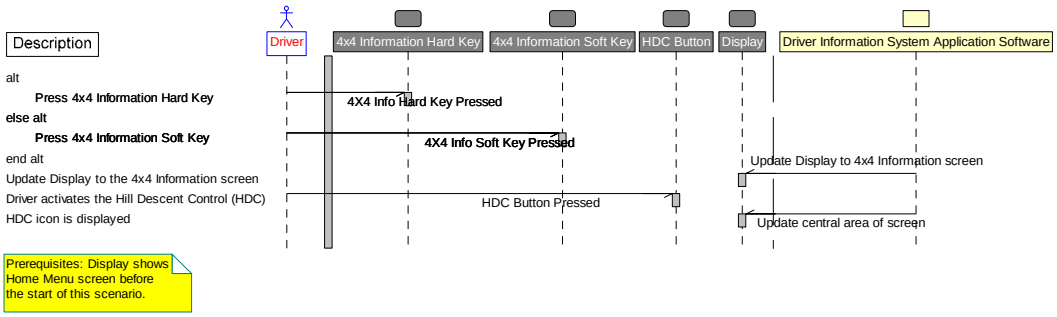


Fig. 4-9. Display of HDC from Home Menu screen.

More scenarios that the 4×4 Information System are required to exhibit are described by other sequence diagrams in the model. Fig. 4-9 is a sequence diagram showing how to display the HDC status from the home menu screen. As shown in Fig. 4-9, as soon as either the 4×4 information hard key or 4×4 information soft key is pressed, the Driver Information System application software will update the display to the 4×4 information screen. The HDC status is displayed by different icons in the central area according to the function selected.

4.2.3.3 State machine diagram

State machine diagrams are used to present different modes and the events that cause the transitions between these modes. Therefore, they can model how the parts of the 4×4 Information System deal with actions in detail.

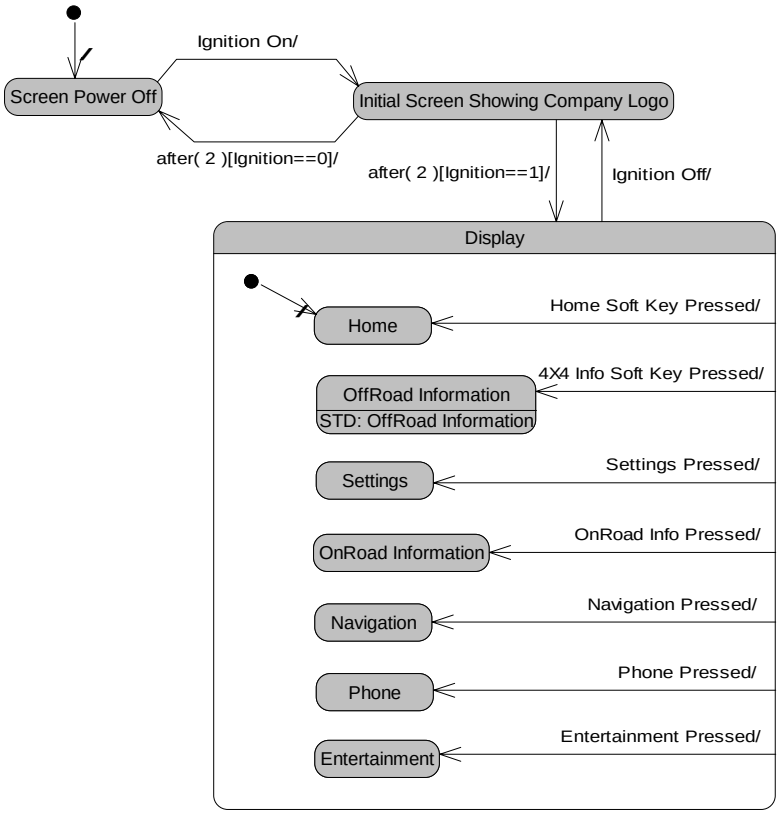


Fig. 4-10. State machine diagram: Driver Information System.

Eight state machine diagrams are utilized to describe the function of the 4×4 Information System. As shown in Fig. 4-10, the first diagram describes the function of the Driver Information System at the highest level and presents information available to the driver in different display modes. The black node on the top left represents the entry point of this state machine diagram. The initial state is “Screen Power Off”. As soon as the ignition is turned on, the screen will display “Initial Screen Showing Company Logo”. After two seconds while the ignition is on, all display modes will be shown on the screen including “Home”, “Off Road Information”, “Settings”, “On Road Information”, “Navigation”, “Phone” and “Entertainment”. The mode can be selected by the driver. The 4×4 information is displayed within the “Off Road Information” mode. The note “STD: Off Road Information” at the bottom of this state indicates entry to a further state machine

diagram at a lower level. They are described in a set of seven separate child state machine diagrams.

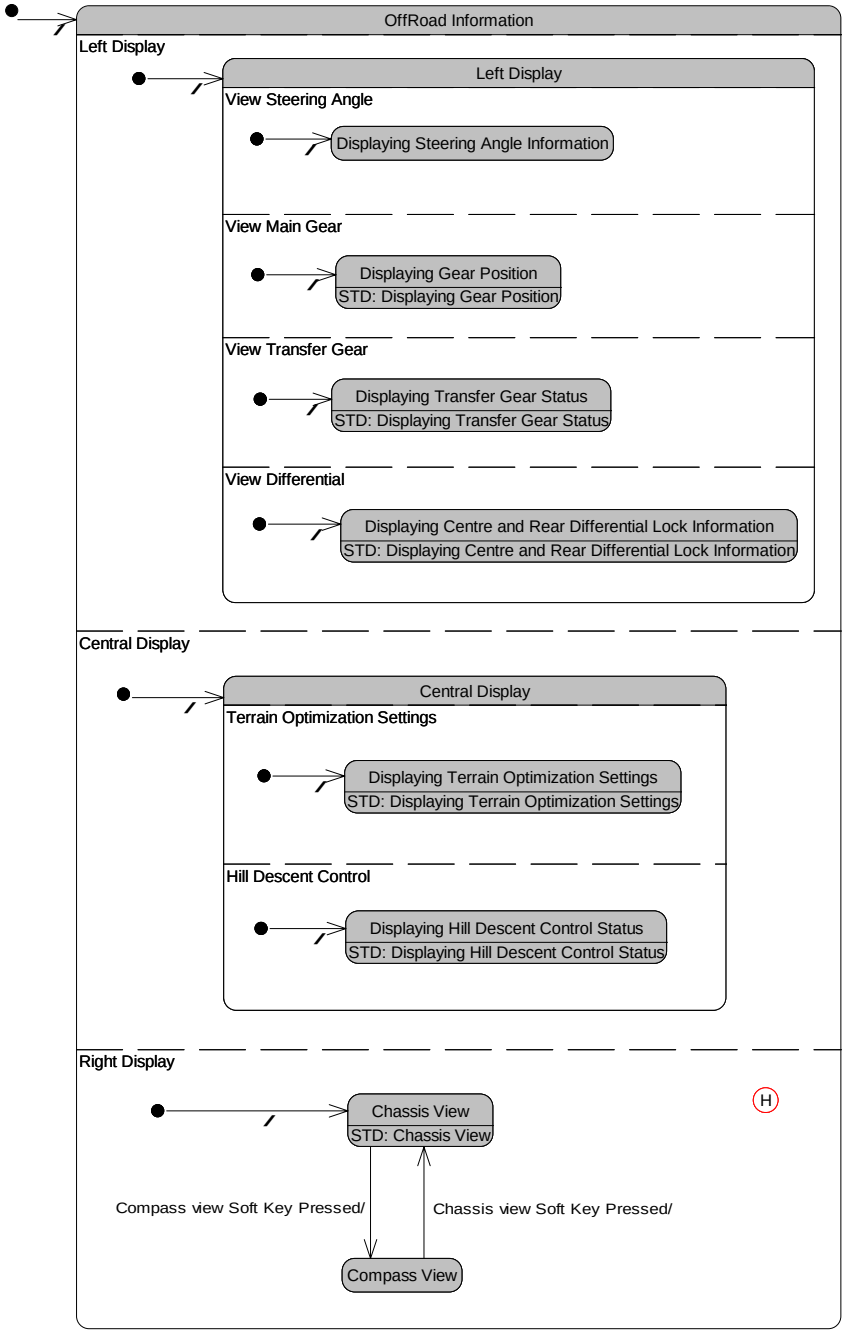


Fig. 4-11. State machine diagram: displaying 4×4 information.

Fig. 4-11 presents how the 4×4 information is distributed in the display. The “Left Display”, “Central Display” and “Right Display”, separated by the dashed line are three concurrent states within the “Off Road Information” state which is

currently displayed. They are shown to the driver at the same time. On the left display, there are four concurrent states showing steering angle information, gear position, transfer gear status, centre and rear differential lock information respectively. The central display shows the TO settings and the HDC status concurrently. The right display can be navigated to “Chassis View” or “Compass View” dependent on the driver’s selection. The “H” within the circle stands for the history state. It indicates that when the display switches back to the off-road information mode from other modes, the right display will navigate back to the previously selected view rather than a default mode.

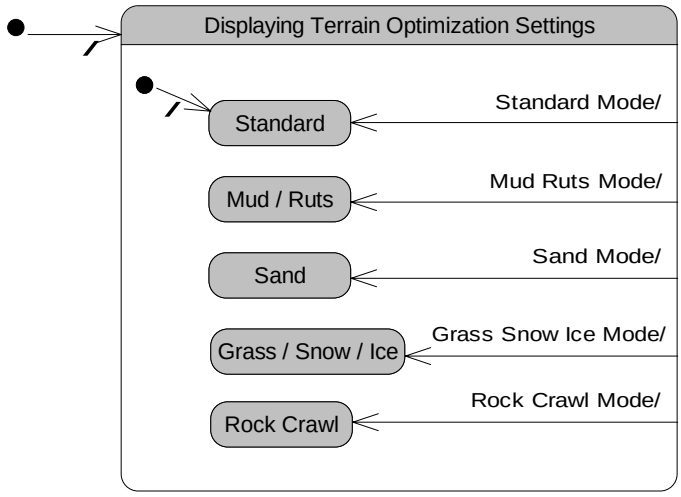


Fig. 4-12. State machine diagram: displaying terrain optimization settings.

The “State machine diagram: Displaying Terrain Optimization Settings”, shown in Fig. 4-12, is developed to detail further the “Displaying Terrain Optimization Settings” state shown in the middle of Fig. 4-11. In Fig. 4-12 the black node at the top left represents the entry point of this state machine diagram. One of five TO settings is displayed when the associated TO mode is selected which is marked on the arrowed lines.

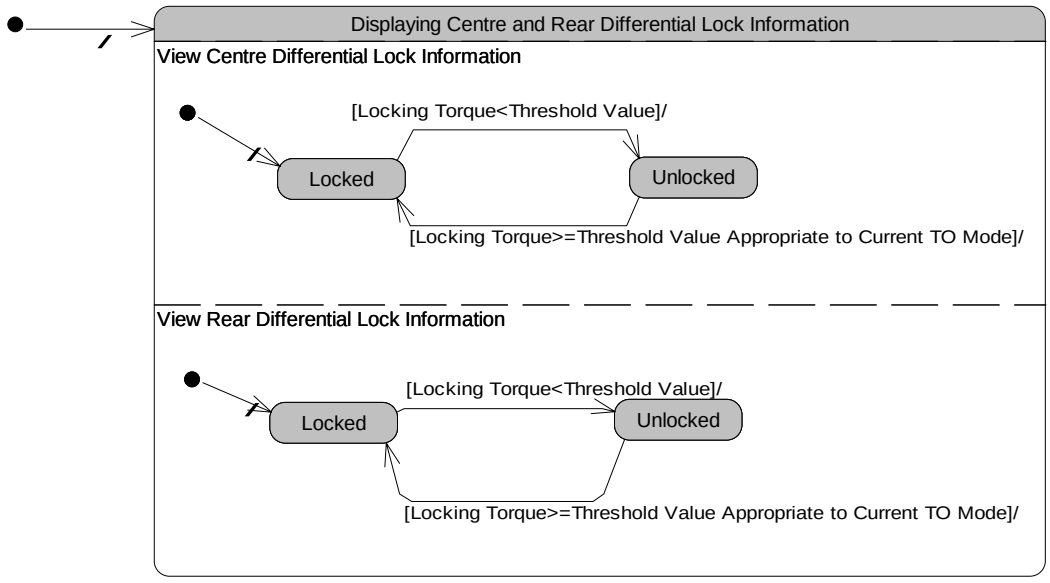


Fig. 4-13. State machine diagram: displaying centre and rear differential lock information.

Fig. 4-13 presents the mechanism of centre and rear differential lock of the 4×4 Information System. “View Centre Differential Lock Information” and “View Rear Differential Lock Information”, separated by the dashed line are concurrent states. They are displayed to the driver at the same time. The mechanism is indicated on the arrowed lines. When the locking torque is less than a threshold value, both centre and rear differential lock show “Unlocked”. They will be locked when the locking torque is higher than the threshold value appropriate to the current TO mode.

4.2.3.4 Activity diagram

The final behavioural diagram examined is the activity diagram. Five activity diagrams are used to model actions and the effect of those actions on relevant artefacts in the model. For example, an activity diagram may describe a use case, event or operation. The top level activities can own an activity diagram, which can describe the detail of that activity in terms of a sequence of actions that may include the use of other, lower level activities. The activity diagram can also

indicate at what point specific inputs are required by actions and specific outputs are produced by them.

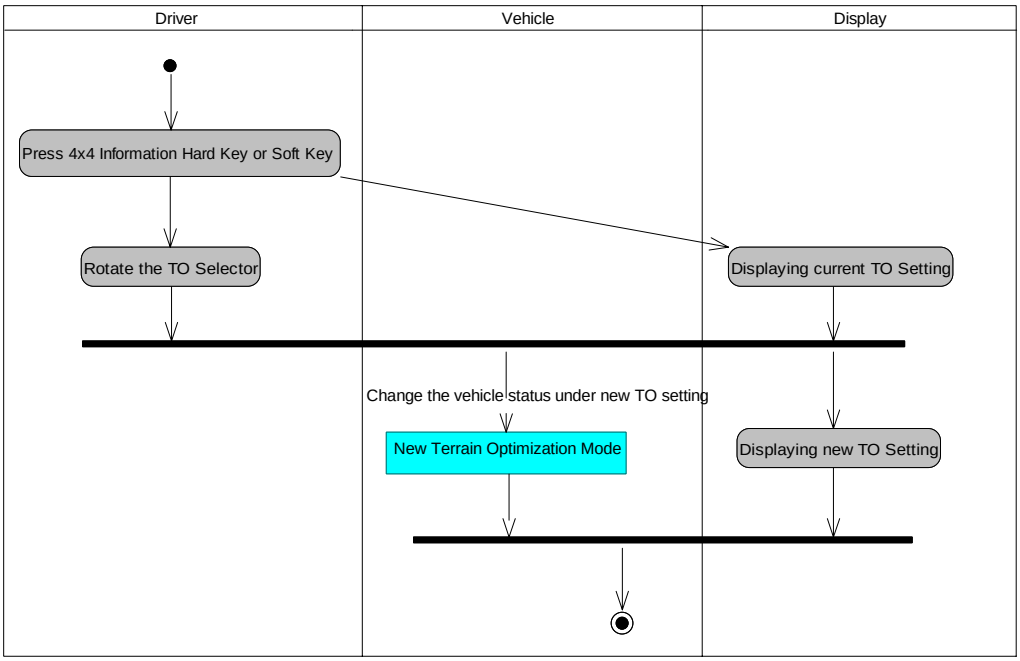


Fig. 4-14. Activity diagram: view TO settings and change the TO mode.

Fig. 4-14 shows an example activity diagram within the 4×4 Information System. Three swim lanes are allocated separately to different objects in the 4×4 Information System. They are “Driver”, “Vehicle” and “Display”. Each object is responsible for performing the activities in its allocated swim lane. As shown in Fig. 4-14, the black node at the top left of this diagram represents the entry point of this activity diagram. The driver performs the action of “Press 4×4 Information Hard Key or Soft Key” to view the 4×4 information on the display and then “Rotate the TO Selector” to change the TO setting. The display will become “Displaying current TO Setting” in response to the driver’s action. The vehicle changes to the selected TO mode while the display shows new TO settings. The black node within the circle at the bottom of this diagram shows the end of this activity diagram.

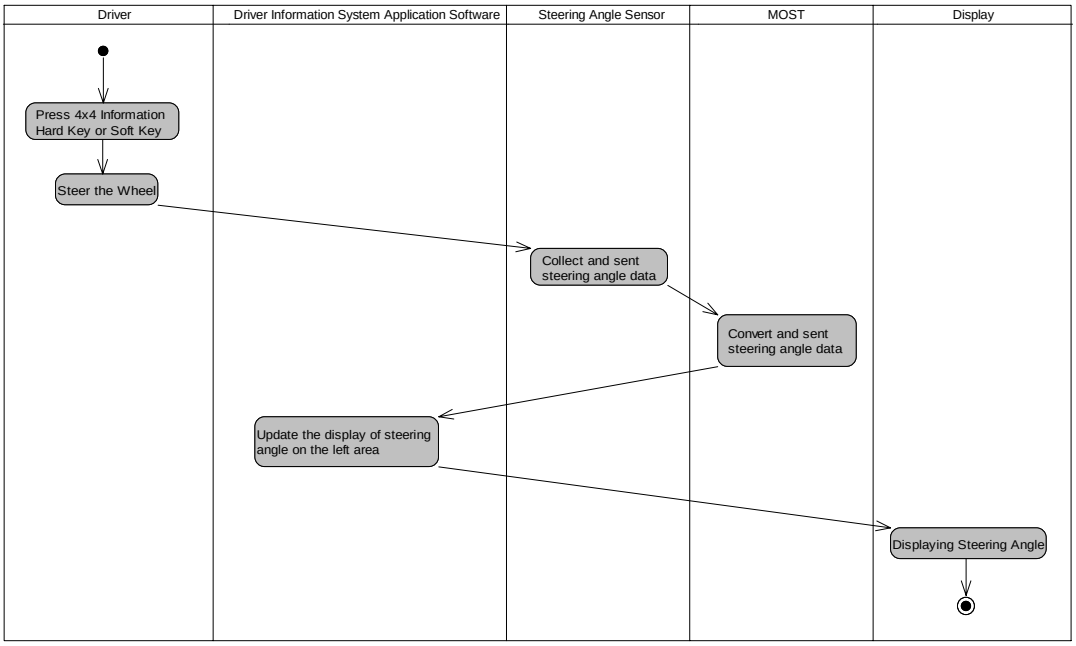


Fig. 4-15. Activity diagram: view steering angle.

Fig. 4-15 is another activity diagram showing actions on relevant artefacts when the driver chooses to view the steering angle. The black node at the top left of this activity diagram represents the entry point. The driver performs the action of "Press 4×4 Information Hard Key or Soft Key" to navigate to the 4×4 information screen and then steer the wheel. The steering angle sensor will collect the steering angle data and send them to the MOST bus. The MOST bus converts the data and delivers them to the Driver Information System application software. The software updates the steering angle data on the left area of the display and the display finally shows the steering angle to the driver.

4.2.4 Other diagrams in the model

The text diagram is not a standard SysML diagram type shown in Fig. 2-15. It can be added to the model as additional notes to illustrate the design requirement of the model. In this model, two text diagrams as shown in Fig. 4-16 and Fig. 4-17 are developed to describe the layout of the display and how the air suspension selector works respectively.

Text Diagram 1.

Description of the Display layout

There are three areas in the driver information system display, left, central and right. The left area is a permanent display area which display chassis and power train of the vehicle with the following information list below:

Steering Angle Information,
High/Low ratio selection status,
Differential lock Information for both centre and rear,
Gear Position.

The central area contains the Home soft key and a permanent display area which shows the following information:

Terrain Optimization Mode,
HDC information.

The right display area changes when different views are selected by press soft keys at the bottom including Compass view and Chassis view.

Fig. 4-16. Text diagram: layout of the display.

Text Diagram 2.

Air suspension selector

Air suspension selector is next to the Hill Descent Control button. The air suspension status can be controlled manually by pushing upwards or downwards the air suspension selector.

Fig. 4-17. Text diagram: air suspension selector.

4.2.5 How diagrams fit together in the model

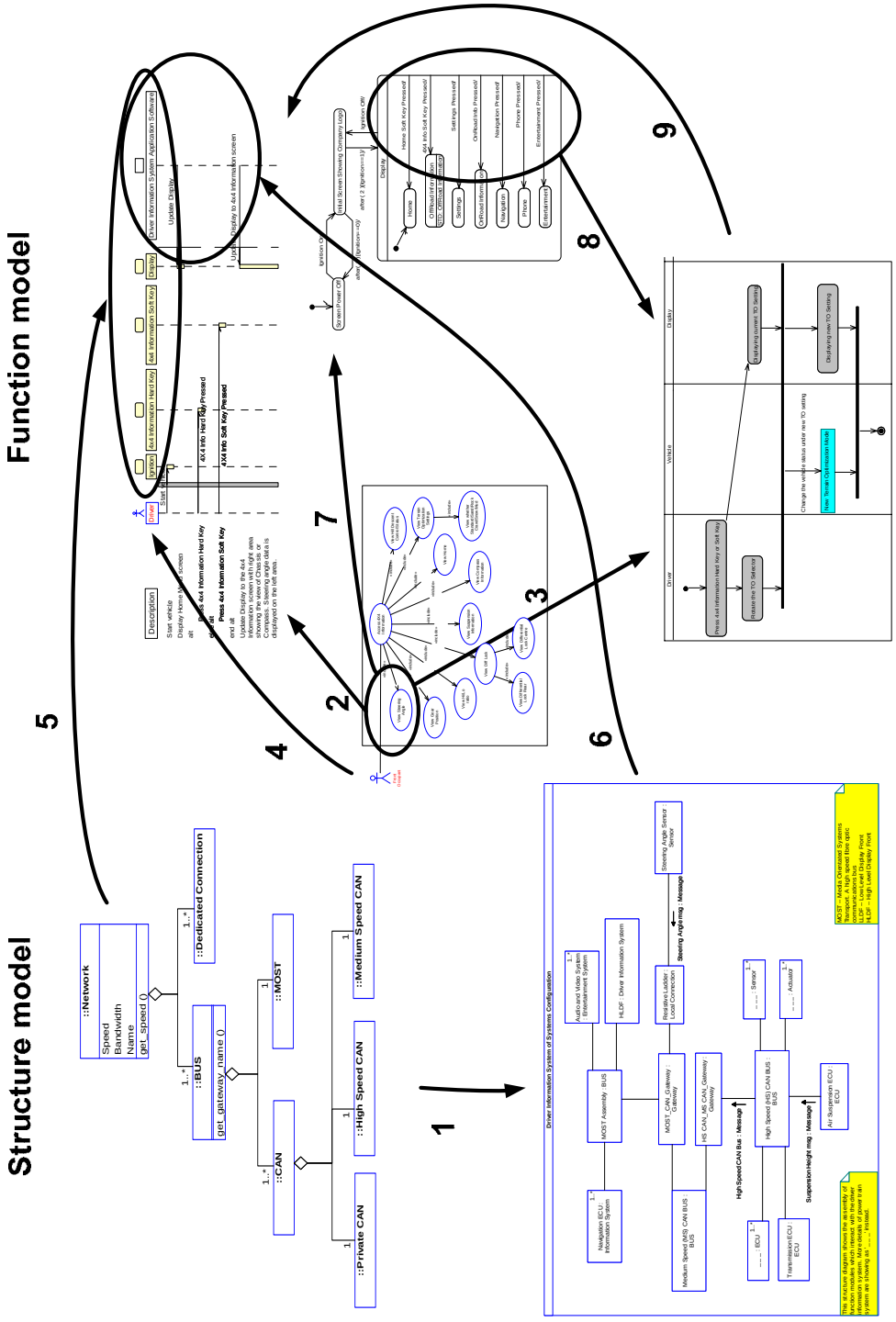


Fig. 4-18. Interactions among diagrams in the model.

Fig. 4-18 shows an example of how to make use of those key diagram types to enable automotive engineers to model both the structure and functionality of an automotive electronic system. This example identifies some of the key relations among the different diagrams in the model. In order to summarize how this model could benefit the engineer during system development, an example requirement of "Driver view steering angle in the 4×4 Information System" is taken. Firstly, block definition diagrams as shown at the top left are used to depict the properties and relations of various data buses which are used for the data transfer in the system. Afterwards, the internal block diagrams are developed to define the internal structure like the connections and data flow direction between those buses. This process is marked as "1". Then the functional modelling has been carried out via use case diagrams, sequence diagrams, state machine diagrams and activity diagrams. Use case diagrams define this "view steering angle" requirement as one use case which is shown in the middle of this diagram. This use case is then modelled in detail within corresponding sequence diagrams and activity diagrams which are marked as "2" and "3" respectively. The sequence diagrams show the interactions among various objects in the system such as how steering angle data are captured and delivered and displayed in this Driver Information System. The actors in the sequence diagram are defined in use case diagrams and are marked as "4". The interface devices and physical objects shown at the top of sequence diagram are modelled in block definition diagrams and internal block diagrams which are marked as "5" and "6". Next, the arrowed line marked as "7" shows state machine diagrams representing how the mode changes under actions in order to display the steering angle information to the driver. The actions are modelled in detail together with the behaviour of relevant objects within the system in activity

diagrams that are marked as “8”. These objects are modelled with their interaction in sequence diagrams that are marked as “9” and also in block definition diagrams and internal block diagrams.

4.2.6 Summary of the diagrams

Table 4-5 lists all the diagrams which were developed for the 4×4 Information System. The first column from the left lists all the diagram types in the model. As shown in this table, eight types of diagram are utilized to build the model of the 4×4 Information System including the text diagram, a non-standard SysML diagram type. The middle column indicates the purpose of the diagram type. Block definition diagrams, internal block diagrams and text diagrams are developed for structural modelling of the system. Use case diagrams, sequence diagrams, state machine diagrams and activity diagrams are utilized for modelling the functional behaviour of the system. The right hand column numbers every diagram in each diagram type. There are 37 diagrams in total in the model which is built in ARTiSAN Studio. A full list of diagrams in this model can be found in Appendix B.

Table 4-5. List of diagrams in SysML model.

Diagram type	Purpose	Name of diagrams
Block definition diagram	Structure	1. Network class 2. Key class 3. Sensor class 4. Gateway_class 5. Sensor_local connection_interface_class
Internal block diagram	Structure	1. Driver Information System of Systems overview 2. MOST System of Systems overview
Text diagram	Structure	1. Layout of the display 2. Air suspension selector
Use case diagram	Behaviour	1. Driver Information System use case 2. 4×4 information use case 3. Suspension information use case
Sequence diagram	Behaviour	1. Return to home menu screen from other screens (HL - B)

		2. Change to 4×4 Information screen from other screens (HL - B) 3. View steering angle (HL - B) 4. View steering angle (HL-W) 5. View steering angle (DL-B) 6. View steering angle (DL-W) 7. Choose different views in 4×4 information screen from home menu screen (HL-B) 8. View main gear and transfer gear change from home menu screen (HL-B) 9. View TO settings from home menu screen (HL-B) 10. Display of HDC from home menu screen (HL-B) 11. Display of suspension status from home menu screen (HL-B) 12. Display of differential status from home menu screen (HL-B) 13. An off-road driving example (HL-B)
State machine diagram	Behaviour	1. Driver Information System 2. Off-road information 3. Displaying gear position 4. Displaying transfer gear status 5. Displaying centre and rear differential Lock Information 6. Displaying TO settings 7. Displaying HDC status 8. Chassis view
Activity diagram	Behaviour	1. Getting to the 4×4 information screen 2. Selecting access height and viewing of new height information 3. View TO Settings and change the TO mode 4. View steering

The listed diagrams in the model provide a clearly structured visualization of the 4×4 Information System. Several types of diagram represent the function requirements from different aspects of the system with relevant objects. Table 4-6 summarizes how the functions of the 4×4 Information System are modelled within use case, sequence, state machine and activity diagrams. The number under each diagram type column refers to the Table 4-5. For example, “Function 3. Switch between different views” is modelled in use case diagram 2, sequence diagram 7 and state machine diagram 2 which are listed in Table 4-5.

Table 4-6. Functions covered by the diagrams in SysML model.

Function number	Function details	Use case diagram number	Sequence diagram number	State machine diagram number	Activity diagram number
1.	Return to home menu screen		1.	1.	
2.	Change to 4×4 information screen from other screens	1.	2.	1.	1.
3.	Switch between different views	2.	7.	2.	
4.	Display steering angle information	2.	3. 4. 5. 6.	2.	4.
5.	Display transfer gear selection status	2.	8.	2.4.	
6.	Display main gear position	2.	8.	2.3.	
7.	Display differential lock information – centre and rear	2.	12.	2.5.	
8.	Display TO mode	2.	9.	2.6.	3.
9.	Display HDC status	2.	10.	2.7.	
10.	Display air suspension status	2. 3.	11.	8.	2.
11.	Display wheel height status	3.	11.	8.	2.

In summary, block definition diagrams and internal block diagrams are used for structural modelling of different aspects. The block definition diagrams depict the properties and relations of various object classes in the system. The internal block diagrams are used to define the internal structure of the system such as connections and data flows between the parts in the system. The use case diagrams define the interaction between actors and associated various use cases for the components within the system. The sequence diagrams show the interactions among various objects in the system when viewed as a sequence in time. State machine diagrams are used to represent mode changes within the system under actions which can be related to operations in the block definition diagrams. The activity diagrams are

used to model the behaviour of relevant objects within the system under the operations defined in the block definition diagrams. A more detailed discussion of the model is provided in Section 4.4 after the functional model built in Simulink/Stateflow is presented in the following section.

4.3 Part of the model developed using MATLAB Simulink/Stateflow

Simulink/Stateflow does not include use case diagrams, sequence diagrams and activity diagrams. It utilizes Stateflow diagrams only to capture the functional behaviour of a system. Fig. 4-19 is the top level view of the functional model of the 4×4 Information System built in Simulink/Stateflow. Compared to Fig. 4-10, there is no distinct difference between them. In this diagram, the "OffRoad_Information" state is highlighted on a grey background. It indicates that this state contains lower level states which are shown in Fig. 4-20.

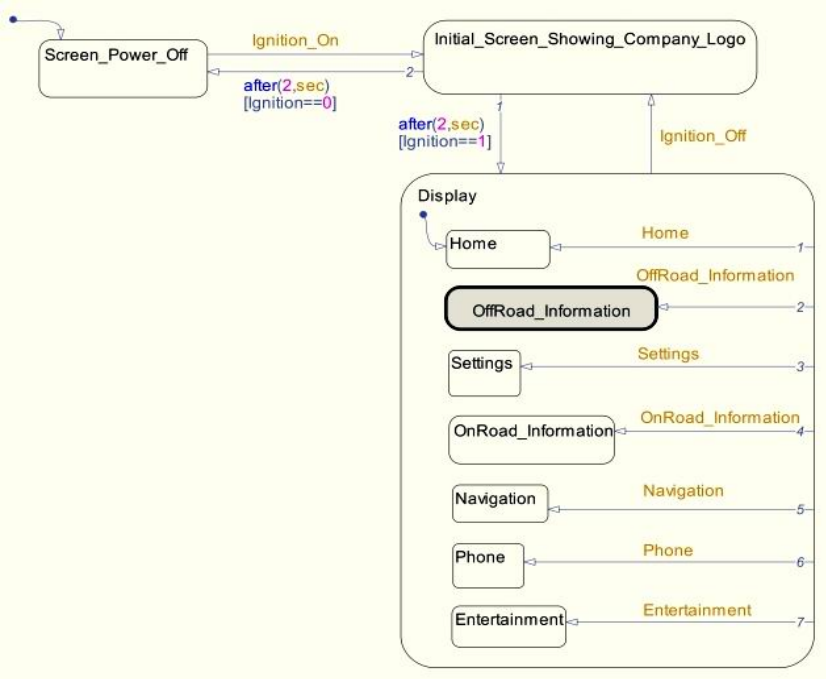


Fig. 4-19. Stateflow diagram: Driver Information System.

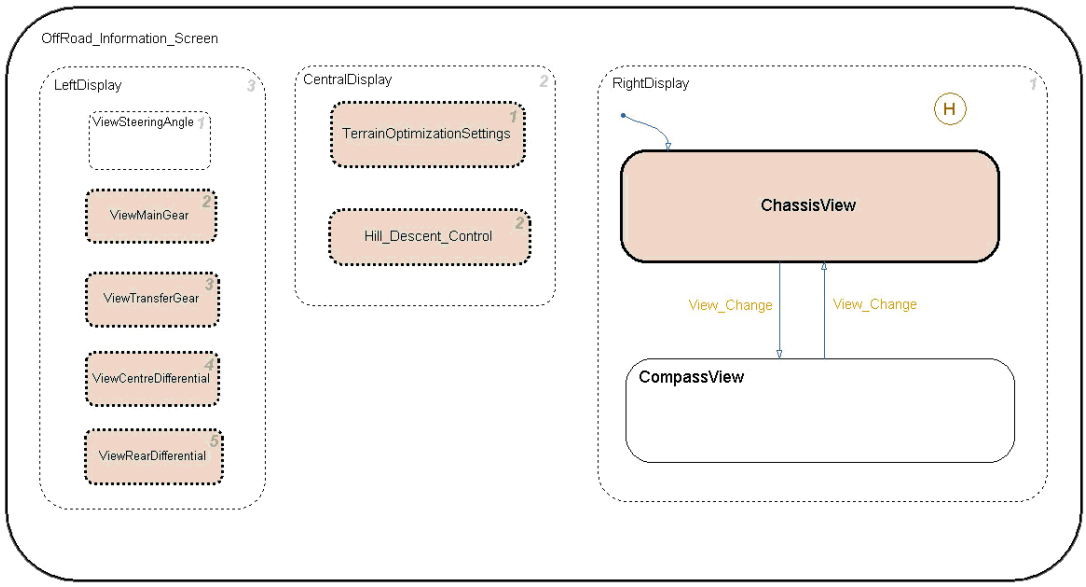


Fig. 4-20. Stateflow diagram: display off-road information.

The 4×4 information displayed in the “OffRoad_Information” mode in Fig. 4-20 is a contrast to the function of the 4×4 Information System modelled and represented in Fig. 4-11. In Fig. 4-20, each rectangle with rounded corners represents a display state. The largest rectangle represents the highest level state in this diagram. Smaller rectangles are lower level states. States in grey and light brown colours contain more detailed level states. Rectangles with dashed lines are utilized to indicate concurrent states. Furthermore, unlike the state machine diagram in ARTiSAN Studio, the location of states in a Stateflow diagram is not restricted. Lower level states can be moved freely within the rectangle of a higher level state. Therefore, the position of each display state in the Simulink/Stateflow model corresponds to the actual HMI. More detailed functionality of the 4×4 Information System is represented by seven separate lower level Stateflow diagrams.

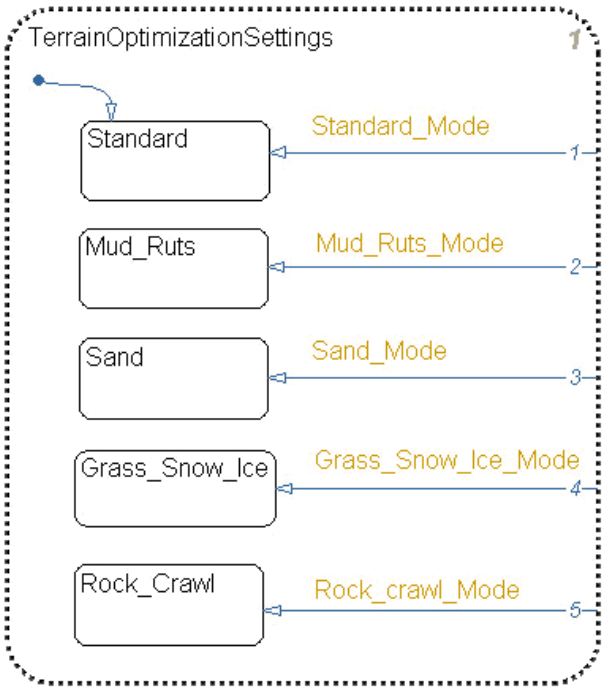


Fig. 4-21. Stateflow diagram: TO settings.

Fig. 4-21 is a lower level state which is shown as a rectangle that is marked with “TerrainOptimizationSettings” in the centre of Fig. 4-20. It is a contrast to the function of the 4×4 Information System modelled and represented in Fig. 4-12. Similar to Fig. 4-12, this diagram represents one of five TO settings is displayed when the associated TO mode is selected, which is marked on the arrowed lines.

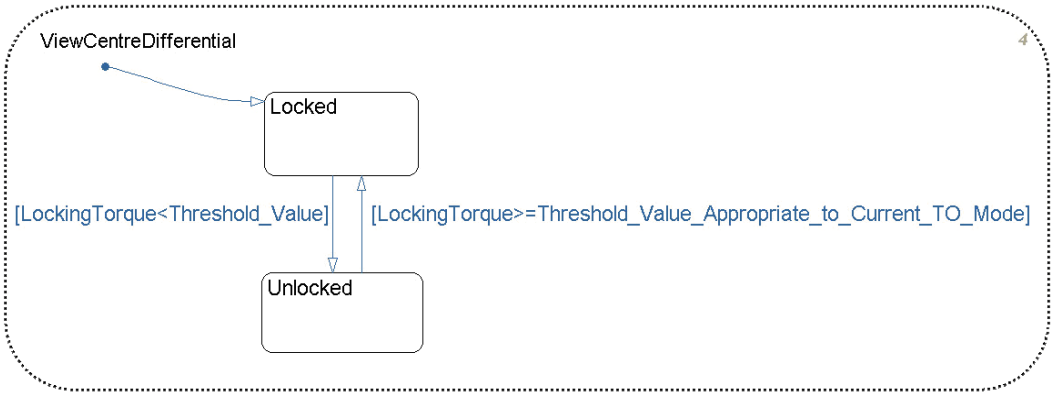


Fig. 4-22. Stateflow diagram: view centre differential.

In contrast to Fig. 4-13, Fig. 4-22 represents the mechanism of the centre differential lock of the 4×4 Information System. When the locking torque is less

than a threshold value, both centre and rear differential locks show “Unlocked”. They will be locked when the locking torque is higher than the threshold value appropriate to the current TO mode.

Fig. 4-23 shows an overview of the Simulink model. All Stateflow diagrams in the model are included in a state chart which is shown as a rectangle at the top right of this figure. Four constant blocks on the left of the state chart are used to model the input value such as “Locking Torque” in the model. Other constant blocks produce control signals to trigger the events which control the transition of states. A vertical bar and three flat bars are “Mux” in the Simulink model which integrate all the control signals and input them through a port at the top of state chart.

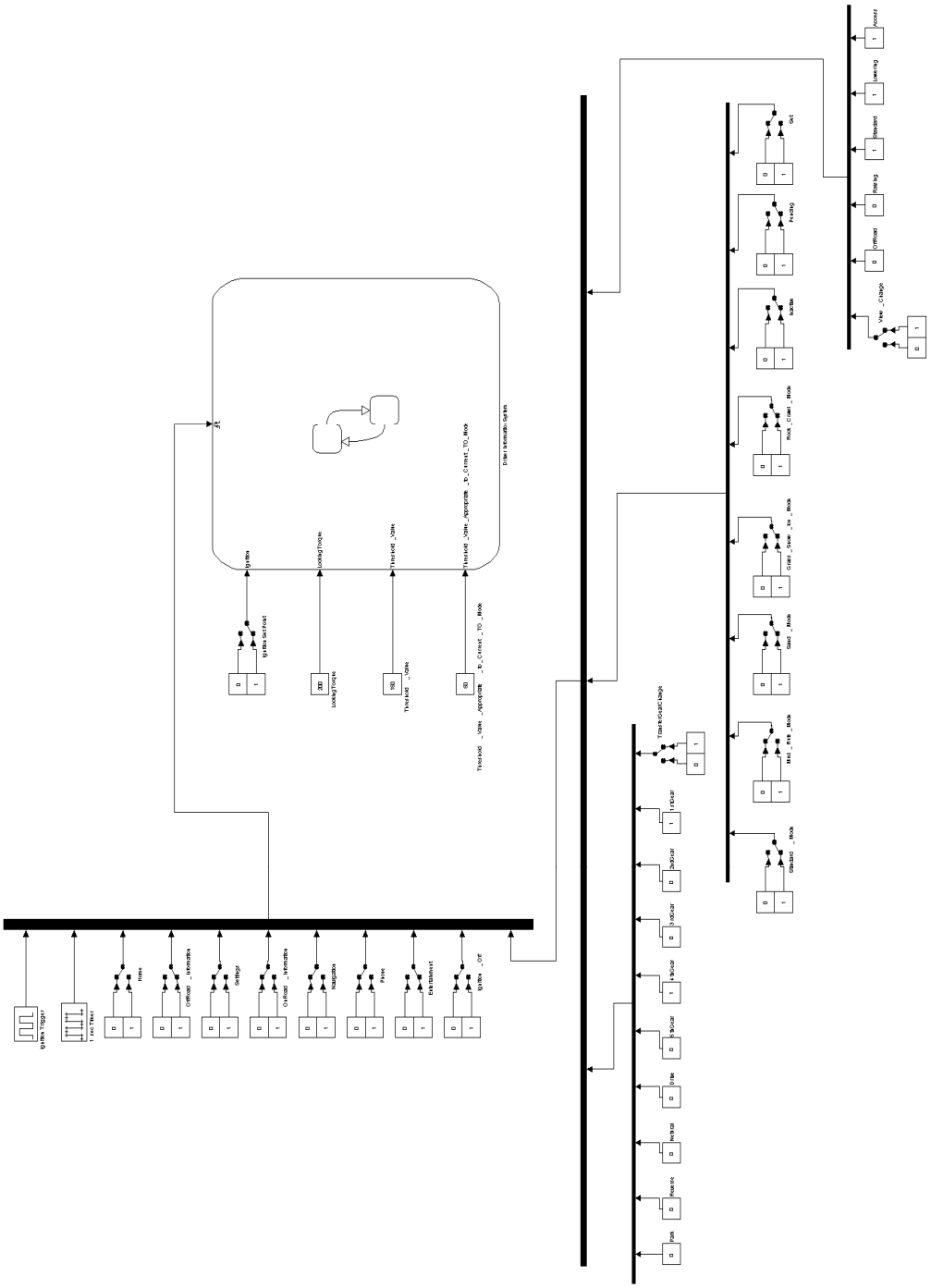


Fig. 4-23. Simulink/Stateflow model of the 4×4 Information System.

Table 4-7. List of diagrams in Simulink/Stateflow model.

Diagram Type	Purpose	Name of Diagrams
Stateflow diagram	Behaviour	1. Driver Information System 2. Display off-road information 3. View main gear 4. View transfer gear 5. View centre differential lock 6. View rear differential lock 7. View TO settings 8. View HDC status 9. Chassis view

Table 4-8. Functions covered by Stateflow diagrams in Simulink/Stateflow model.

Function Number	Function Details	Stateflow Diagram Number
1.	Return to home menu screen	1.
2.	Change to 4×4 information screen from other screens	1.
3.	Switch between different views	2.
4.	Display steering angle information	2.
5.	Display transfer gear selection status	2.4.
6.	Display main gear position	2.3.
7.	Display differential lock information – centre and rear	2.5.6.
8.	Display TO mode	2.7.
9.	Display HDC status	2.8.
10.	Display air suspension status	2.9.
11.	Display wheel height status	2.9.

Table 4-7 lists all the Stateflow diagrams which were developed for the 4×4 Information System. As shown in this table, the Simulink/Stateflow model utilizes Stateflow diagrams only for modelling the functional behaviour of the system.

Every diagram is numbered in the right hand column. There are nine Stateflow diagrams in total in this model. A full list of Stateflow diagrams in the model can be found in Appendix C. Table 4-8 summarizes how the functions of the 4×4 Information System are modelled within these Stateflow diagrams. The Stateflow diagram number refers to Table 4-7. For instance, “Function 5. Display Transfer Gear selection status” is modelled in Stateflow diagrams 2 and 4 which are listed in Table 4-7.

4.4 Discussion

In this chapter, diagrams have been selected from the full set of the SysML diagram types as being representative of a typical system development. This chapter has discussed how to make use of those key diagram types to enable automotive engineers to model both the structure and function of an automotive electronic system.

Block definition diagrams and internal block diagrams are used for structural modelling of different aspects. The block definition diagrams depict the properties and relations of various object classes in the system. The internal block diagrams are used to define the internal structure of the system such as connections and data flows between the parts in the system.

The functional modelling has been carried out via use case diagrams, sequence diagrams, state machine diagrams and activity diagrams. The use case diagrams define the interaction between actors and associated various use cases for the components within the system. The sequence diagrams show the interactions among various objects in the system when viewed as a sequence in time. State machine diagrams are used to represent mode changes within the system under

actions which can be related to operations in the block definition diagrams. The activity diagrams are used to model the behaviour of relevant objects within the system under the operations defined in the block definition diagrams.

As described in Section 4.2.5, the diagrams fit together as one model, there is no obvious integration or data synchronization issue that needs to be considered explicitly when additional diagrams are introduced into the model. Facilitated by ARTiSAN Studio, the entire model is synchronized by keeping attributes of the same component in the model consistently updated among all diagrams. For example, if an actor's name needs to be changed due to the alteration of the requirement document, revising the name in any diagram that includes this actor results in the name alteration in all diagrams that contain this actor. Thus, all diagrams of the model are internally synchronized with external requirements.

Getting the requirements right is crucial for any project [80, 81]. These diagrams facilitate the formulation of textual form specification documents by successfully translating the design requirements into a clearly structured and visualized model. This model represents the design requirement in both structural and behavioural viewpoints. Moreover, experience shows that these diagrams can be broken down and developed at different levels to represent the interactions and detail at various levels. The integration is vital for the automotive electronic system. Different people require different pieces of information, depending on what their roles are in the system. This model facilitates collaboration of people with different backgrounds such as systems engineers and software engineers to design, implement and manage such complex distributed systems and integrate them into one cohesive and reliable SoS. Specifically, the systems engineer can make use of block definition diagrams and internal block diagrams to define the physical

structure and capture the interaction between the components. The software engineer can pay more attention to the state machine diagrams, activity diagrams and sequence diagrams for the functional modelling and then deliver the software. The SysML can benefit the development of an automotive electronic SoS through either way [116, 117].

The function model has also been developed in Simulink/Stateflow. In the Simulink/Stateflow model, a Stateflow diagram has a very similar mechanism as a state machine diagram in SysML to model the functionality of the 4×4 Information System. Diagrams can be broken down and developed at different levels to show the interaction and detail at various levels such as Fig. 4-19 and Fig. 4-20. How mode changes under actions in order to display the 4×4 information to the driver are represented in a similar mechanism as state machine diagram in SysML. The developer can gain an understanding of the system from several levels. However, for the purpose of analysing a system, it is important to be able to observe a system from many different viewpoints [27]. In comparison with the SysML model, the Simulink/Stateflow model only utilizes nine Stateflow diagrams to provide the description of the function view of the 4×4 Information System from the behavioural viewpoint. The model built in SysML benefits from the collaboration of six diagram types, namely, block definition diagram, internal block diagram, use case diagram, sequence diagram, state machine diagram and activity diagram to enable the engineer to obtain a profound understanding of the SoS from both structural viewpoint and behavioural viewpoint.

It is clear that SysML provides a more complete description of the functional behaviour of the SoS than Simulink/Stateflow. The added detail in SysML provides

a more comprehensive coverage of the functional behaviour of the SoS in relation to the design requirements.

An important consideration for model construction is the capability of coding implementation. The code generated automatically through model building in ARTiSAN Studio and the model developed in Simulink/Stateflow are going to be described in the following chapter.

Chapter 5

Code generation and verification

Chapter 4 presented the development of the structure and function model in SysML by using ARTiSAN Studio and the building of the function model in Simulink/Stateflow. This chapter compares the functional equivalence of the function models and discusses the automatic code generation from both models. The PolySpace tool is utilized to perform automatic code verification for the C code generated. The attention focuses on the comparison of quality and efficiency of the code.

5.1 Function model simulation and comparison

Utilizing simulation to verify the model is a sophisticated task for model-based design. For example, the control engineer needs to transform design requirements into the block diagrams in a Simulink model. After completing the model, offline simulation can be performed for the system analysis. In this chapter, eleven test cases are developed as shown in Table 5-1 to verify the functional equivalence of the state machine diagrams in the function model built in ARTiSAN Studio and Stateflow diagrams in Simulink/Stateflow.

Table 5-1. Test cases for verifying the functional equivalence of the model.

Number	Test case name	Precondition	Objective
1.	Return to home menu screen	Display shows 4×4 information screen	Display home menu screen as soon as home menu hard key or soft key is pressed.
2.	Change to 4×4 information screen from other screens	Display shows home menu screen	Display 4×4 information screen as soon as the 4×4 information hard key or soft key is pressed.
3.	Display compass information	Display shows home menu screen	Display compass view on 4×4 information screen.
4.	Display steering angle information	Display shows home menu screen.	Display steering angle information on 4×4 information screen.
5.	Display transfer gear selection status	Display shows home menu screen	Displaying the transfer gear status when a High or Low ratio has been selected.
6.	Display main gear position	Display shows home menu screen	Displaying the main gear position when main gear position has been selected.
7.	Display differential lock information – centre and rear	Display shows home menu screen	Displaying the differential lock information for both centre and rear differentials on 4×4 information screen.
8.	Display TO mode	Display shows home menu screen	Displaying five different driving modes according to the driver’s selection.
9.	Display HDC status	Display shows home menu screen	Displaying the HDC status based on the driver’s selection.
10.	Display air suspension status	Display shows home menu screen	Displaying the air suspension status when chassis view is selected in the 4×4 information screen.
11.	Display wheel height status	Display shows home menu screen	Displaying the wheel height status when chassis view is selected in the 4×4 information screen.

The simulation is performed in both tools for these test cases. During the simulation, the status of states changes according to the input actions. Thus, the control and supervisory logic which are defined in the function models can be tested by using the above test cases. For example, test case 3 is used to verify the function of displaying the compass view in the 4×4 Information System. During the simulation, both tools simulate state changes in the function model based on the same set of actions which is described as the main flow in the Table 4-4. When the test begins, both models show the home menu screen during the simulation, after the actions are taken, the screenshots of the simulations in both models are shown below.

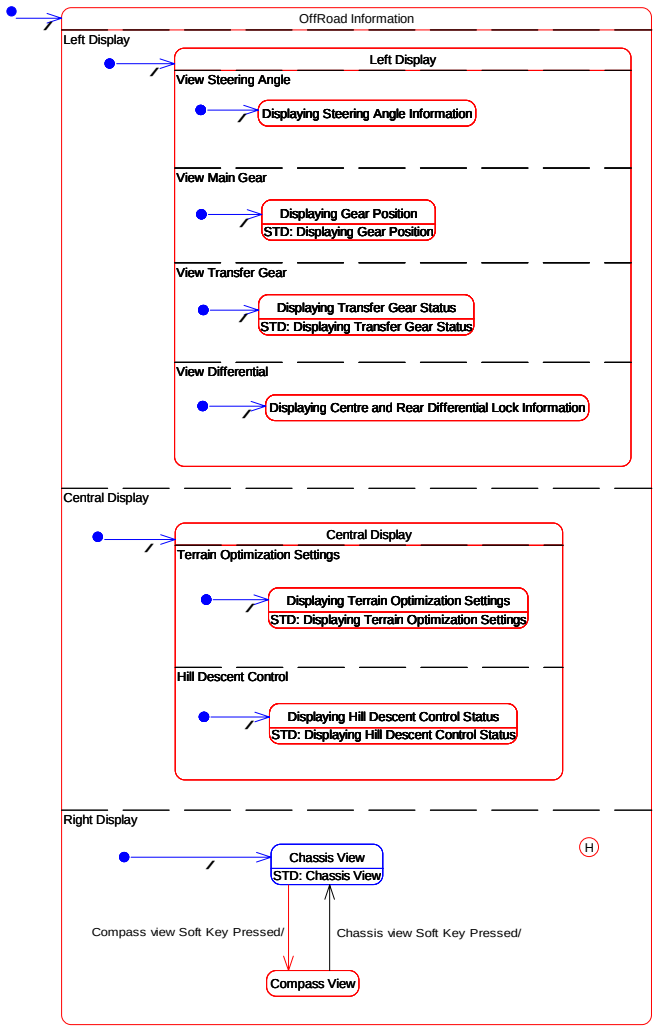


Fig. 5-1. Simulation in ARTiSAN Studio - OffRoad Information.

Fig. 5-1 shows the simulation performed in ARTiSAN Studio. It simulates the 4×4 Information System currently displaying “OffRoad Information” mode. The active states are represented in red during the simulation. Fig. 5-1 shows the left area of the screen displays the steering angle, main gear, transfer gear, and centre and rear differential lock. The central display shows the TO settings and HDC. The right area of the screen is in compass view.

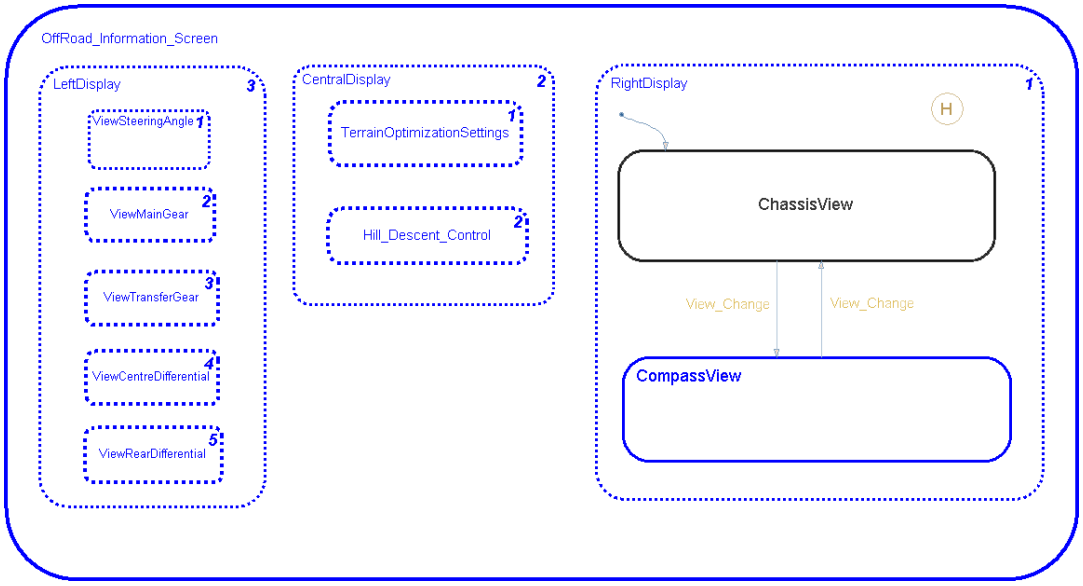


Fig. 5-2. Simulation in Simulink/Stateflow - OffRoad Information.

Within the model built in Simulink/Stateflow for the 4×4 Information System, the simulation is performed as shown in Fig. 5-2. This diagram shows the simulation of the 4×4 Information System which is in comparison with Fig. 5-1. The active states are represented in blue during the simulation. Fig. 5-2 shows that the screen is in “OffRoad Information” mode. The left area of the screen, the central display and the right area of the screen shows the same information as simulated in the state machine diagram in ARTiSAN Studio.

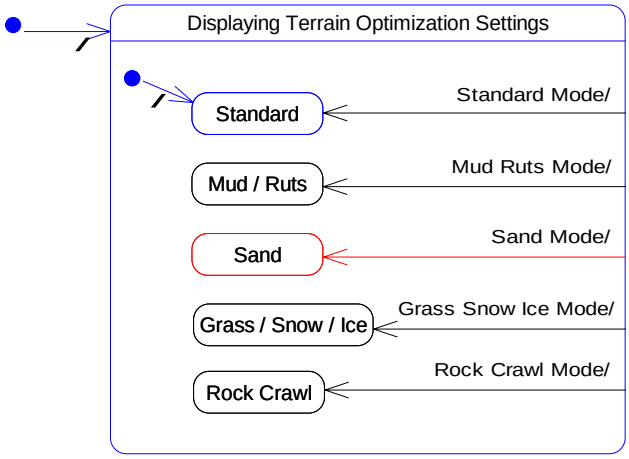


Fig. 5-3. Simulation in ARTiSAN Studio - Display TO mode.

Test case 8, 'Display TO mode' is employed as another example in this section. Fig. 5-3 and Fig. 5-4 present how the function model behaves after the same actions were taken. Fig. 5-3 shows that the sand mode is displayed on the screen which is highlighted in red in the model built by ARTiSAN Studio. Similarly, Fig. 5-4 shows that the screen is showing sand mode which is represented in blue in the model developed in Simulink/Stateflow.

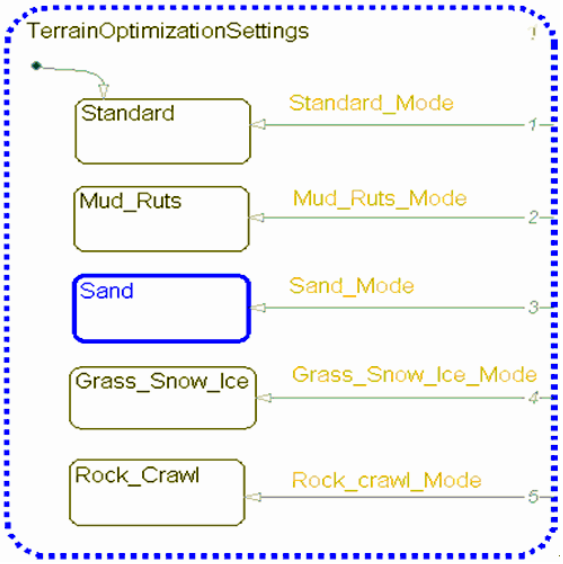


Fig. 5-4. Simulation in Simulink/Stateflow - Display TO mode.

From the comparison of the simulation exercise, it is concluded that the function models built in ARTiSAN Studio and in Simulink/Stateflow perform the same behaviour under the same input action, i.e., they are functionally equivalent. The code generation from the functional models is explored in the next section.

5.2 Code generation

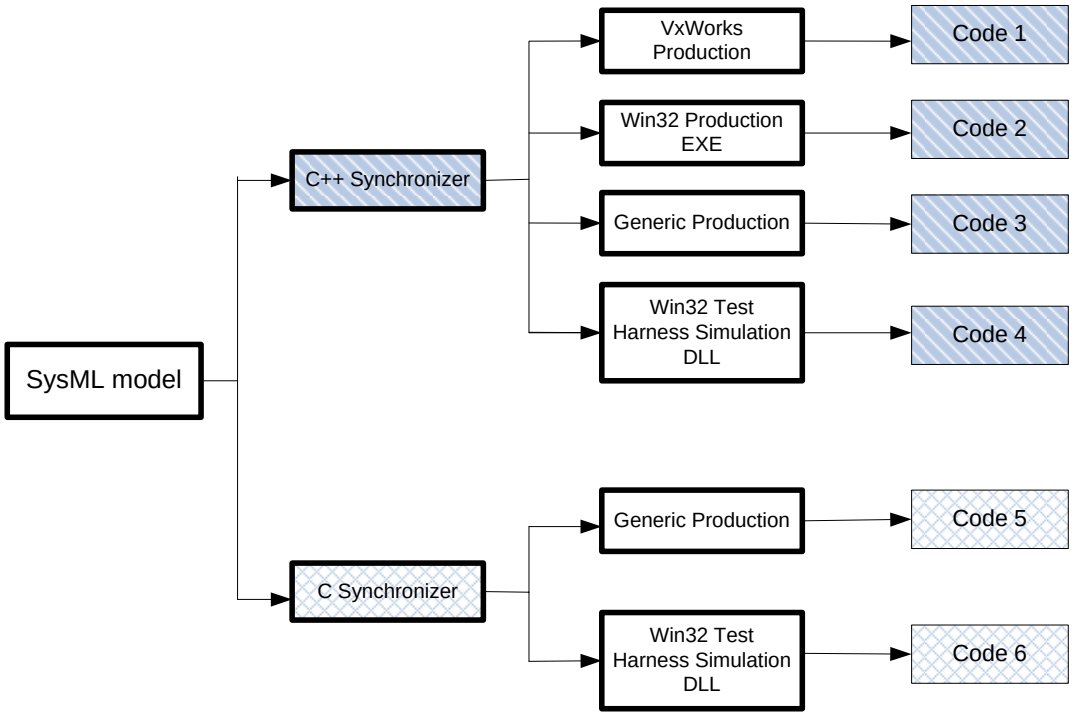


Fig. 5-5. Code generation in ARTiSAN Studio for the model of 4×4 Information System.

The code generation of the function model built in SysML is facilitated by the C Synchronizer and C++ Synchronizer in ARTiSAN Studio version 6.1. The C Synchronizer is a well integrated tool in ARTiSAN Studio. It uses a set of generation templates to produce the code for different purposes which is shown in Fig. 5-5. For example, C code can be generated through Win32 Production EXE scheme for a Windows operating system. C++ code can be produced through the VxWorks Production scheme for a VxWorks operating system. Additionally,

ARTiSAN Studio provides the engineer with a certain level of flexibility for code generation. For instance, the code can be generated from state machine diagrams in the function model or it can be produced for the entire model including both structure and function models. In this Thesis, C code is produced from eight state machine diagrams in the function model through a Generic Production scheme for operating systems that are not Windows or VxWorks. It is indicated as “Code 5” in Fig. 5-5. This C code contains 4,311 lines (or 3,142 lines without comments).

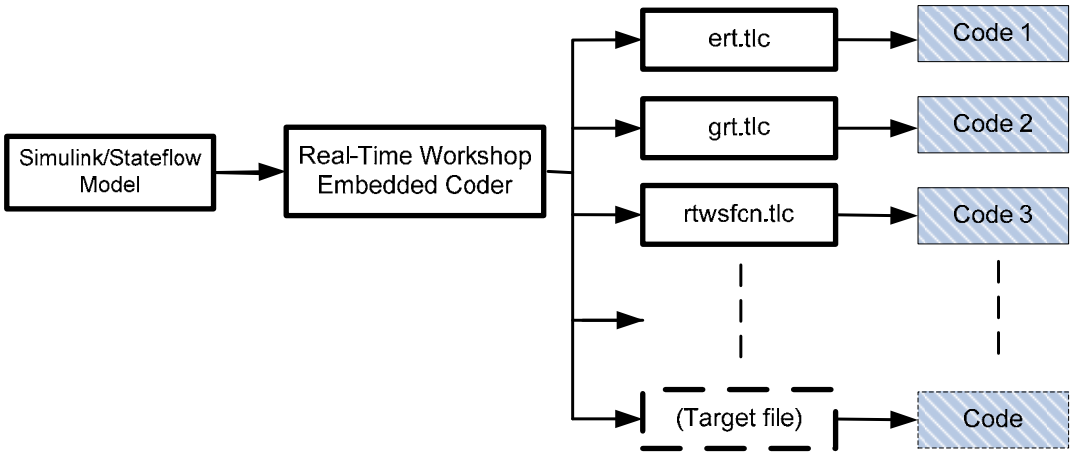


Fig. 5-6. Code generation in Real-Time Workshop Embedded Coder for the model of 4×4 Information System.

The Real-Time Workshop Embedded Coder (RTW EC) is one of many in the Simulink product family which has been developed by The MathWorks. The version evaluated in this Thesis is R2007a. It enables the C code and C++ code generation for the Simulink and Stateflow models. The different target files can be selected for generating C or C++ code, which are shown in Fig. 5-6. In this Thesis, C code is generated from nine Stateflow diagrams in the Simulink/Stateflow model through the “Real-Time Workshop Embedded Coder with no auto configuration” scheme by selecting the “ert.tlc” target file. This C code is PC-based C source code which has a difference lap to the final stage microcontroller-based code. To

automatically generate C source code and run the code directly on the target, some modifications are required to the target files which are provided by RTW EC. The other options are set as the default when the C code is generated. It is indicated as "Code 1" in Fig. 5-6. This C code has 1,818 lines (or 1,336 lines without comments). Fig. 5-7 shows the default parameter configuration of RTW EC.

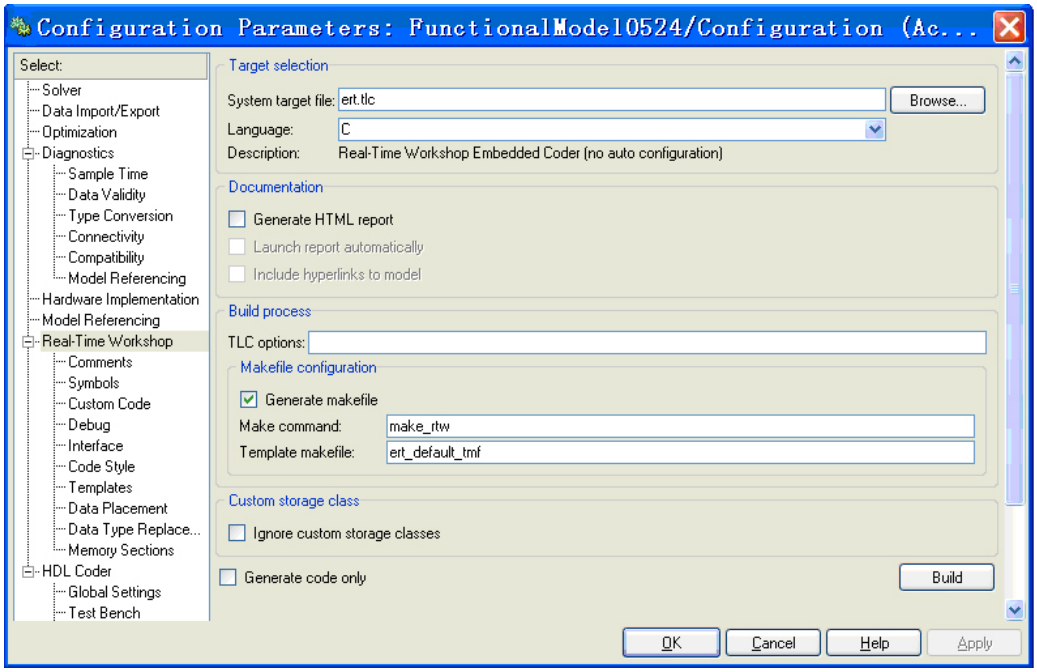


Fig. 5-7. Parameter configuration of Real-Time Workshop Embedded Coder.

Fig. 5-8 and Fig. 5-9 show the default parameter configuration of the solver and optimization respectively.

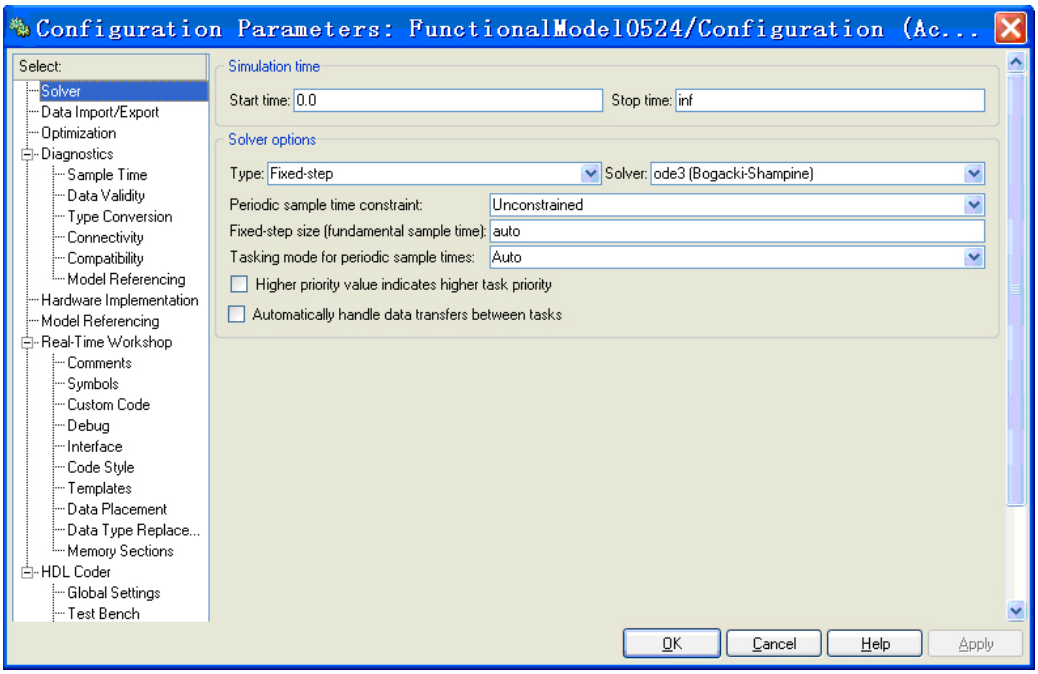


Fig. 5-8. Parameter configuration of solver.

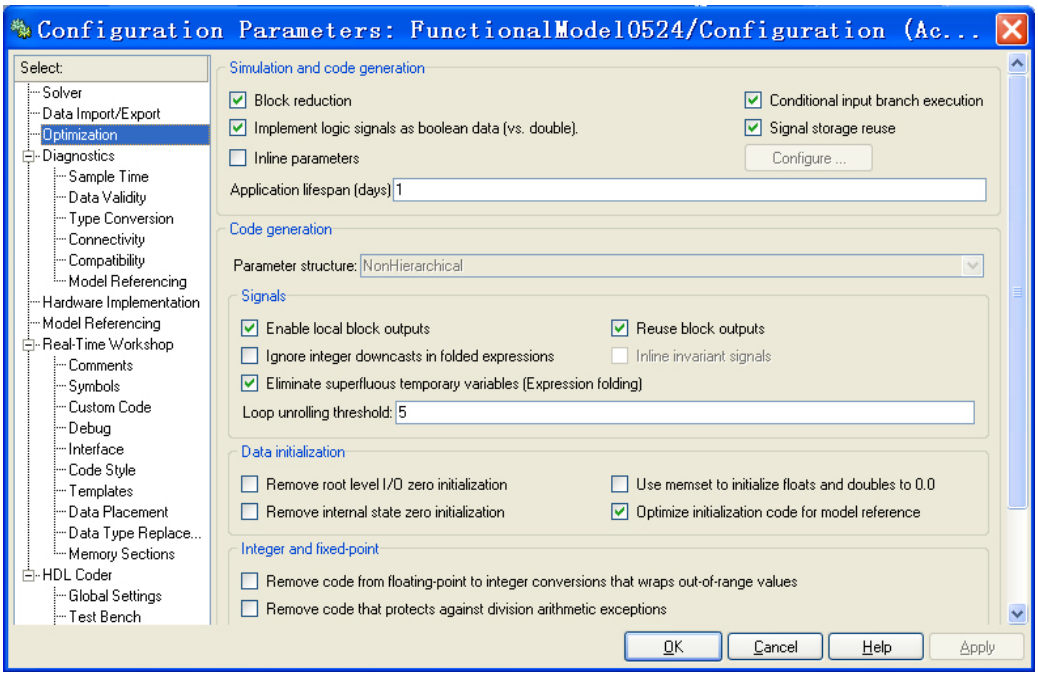


Fig. 5-9. Parameter configuration of optimization.

Further investigation focuses on the comparison of quality and efficiency of the code. Selecting different options in the code generation could potentially affect the result of the code generation in terms of quality and efficiency. Attention is given and the results observed are discussed in a later section.

5.3 Automatic code verification and code comparison

Comparing the length of the code, the C code generated from ARTiSAN Studio for state machine diagrams is twice as long as that produced by Simulink/Statelow. It implies that this code takes a longer time to compile and is more difficult to check and maintain. Ideally, developers would discover and fix errors in programs before they are released [118]. However, it is an extremely difficult task. Among the many approaches for finding and fixing errors, static analysis is one of the most attractive [119-122]. Static analysis aims to automatically process source code and analyze all code without the large amount of test cases used in testing [123]. PolySpace is such a tool to use static analysis techniques, including symbolic analysis, abstract interpretation, model checking, integer range analysis, and inter-procedural analysis [97]. Hence, it is utilized in this Thesis to perform automatic code verification for the C code generated from both models. The PolySpace analysis process is composed of three main phases. Firstly, PolySpace checks the syntax and semantics of the analyzed files. Then, PolySpace seeks the main procedure. If one is not found, PolySpace will generate one automatically. By default, this function will call all public functions of the file. Finally, PolySpace proceeds with the code analysis phase, during which run time errors are detected and highlighted in the code. Moreover, each operation checked is displayed by using a meaningful colour scheme and related diagnostics [97]:

- Red: Errors which occur at every execution.
- Orange: Warning – an error may occur.
- Grey: Shows unreachable code.
- Green: Error condition that will never occur.

As shown in Fig. 5-10, the PolySpace Viewer displays the analysis results of the C code produced from the state machine diagrams of the model built in ARTiSAN Studio. The left hand area of this diagram is the procedural entities view. It shows the list of packages that have been analyzed or used during the analysis. On the right of this window is the source code view with coloured instructions which are stated as above. These windows enable software engineers to easily inspect the source code. For example, when the engineer clicks on “RtsDrive ( )” (marked as “1”), it expands and displays a list of coloured symbols showing the diagnostic results. In the meantime, the source code is opened and displayed on the right hand side in the source code view. After clicking on the orange item “NIV.5” (marked as “2”) which stands for Uninitialized Variable, the source code view is updated to show the location of this orange warning.

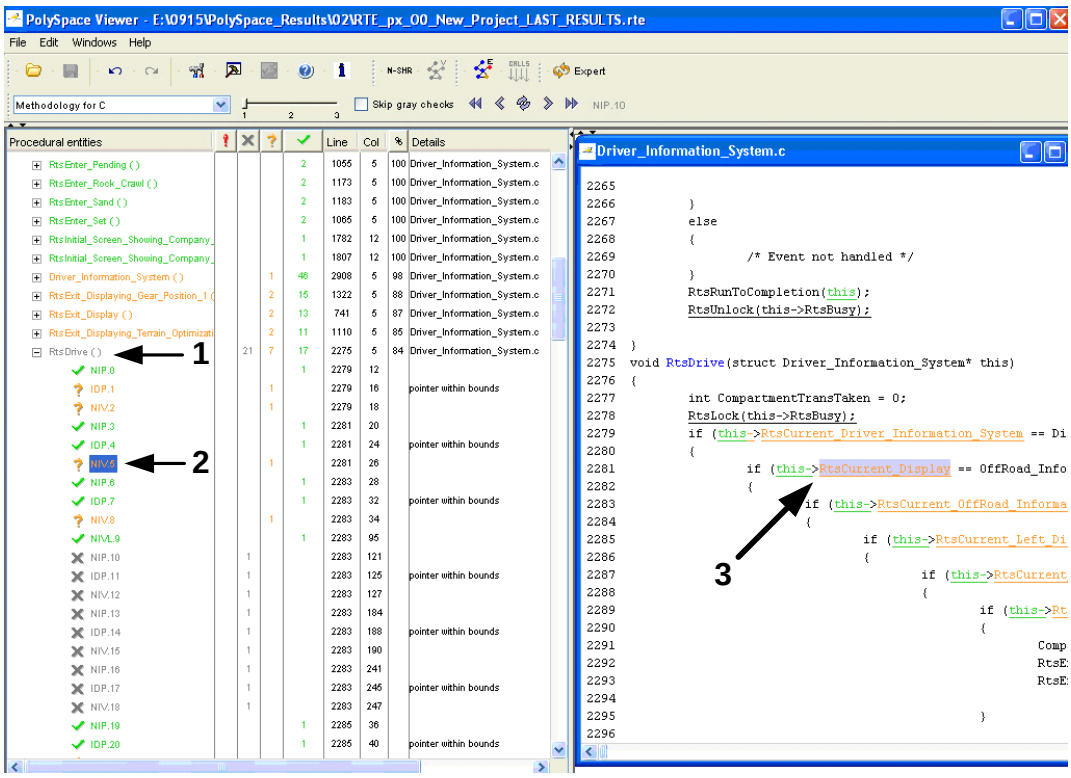


Fig. 5-10. Analysis result of the C code produced from ARTiSAN Studio.

In addition, a warning message window is opened after clicking on the orange section with the grey background (marked as “3”) of the code in Fig. 5-10. Fig. 5-11 presents this warning message window and it precisely indicates line 2281, column 26 of the source code has a variable that may be non-initialized.

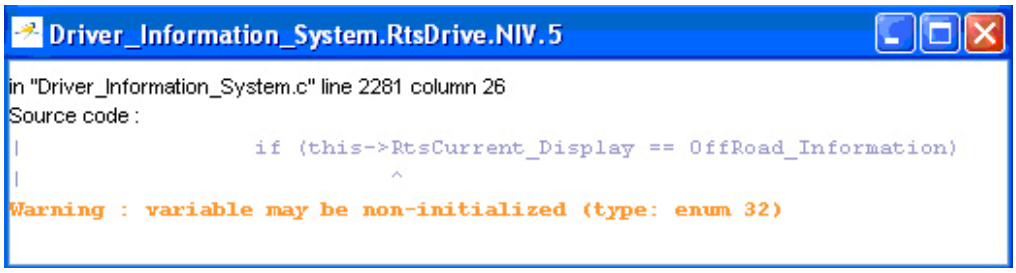


Fig. 5-11. Warning message window of the C code produced from ARTiSAN Studio.

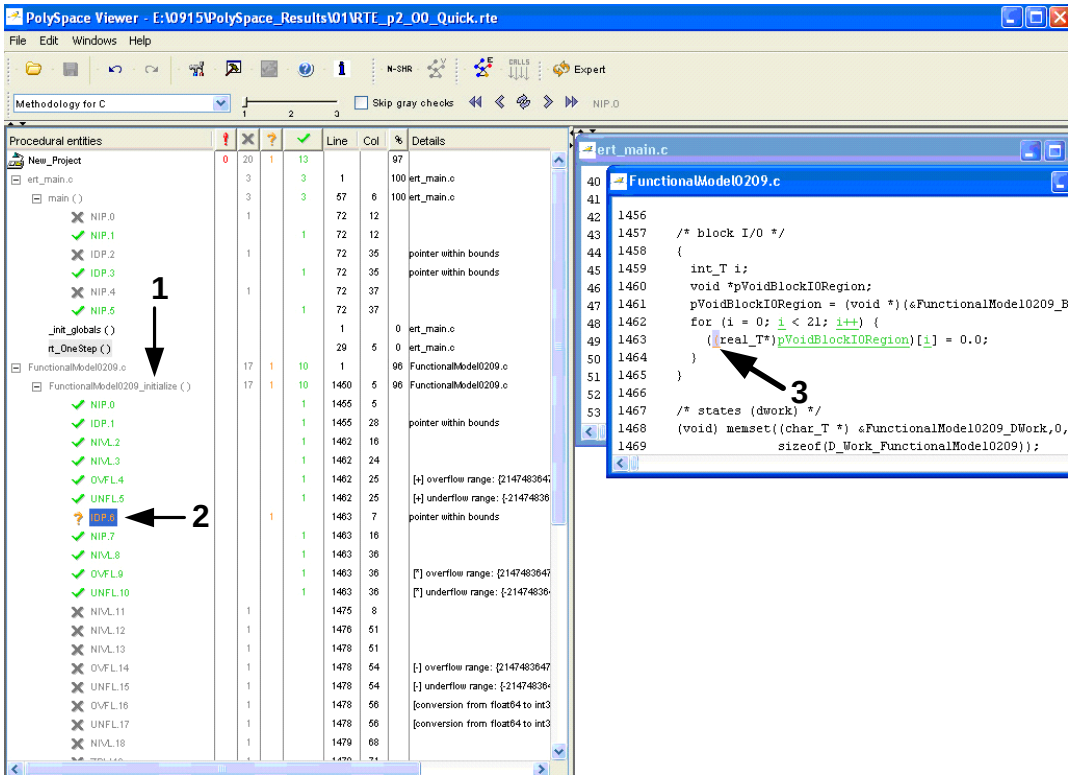


Fig. 5-12. Analysis result of the C code produced from RTW.

Fig. 5-12 represents the PolySpace Viewer showing the analysis results of the C code produced from the Stateflow diagrams in the Simulink/Stateflow model by

RTW EC. On the left of this window is the procedural entities view. It displays the list of packages that have been analyzed or used during the analysis. The right hand area of this window is the source code view with coloured instructions. Software engineers can manually inspect the source code from this window. For instance, when the engineer clicks on “FunctionalModel0209_initialize ( )” (marked as “1”), it expands and displays the diagnostic results in coloured symbols. At the same time, the source code is opened and exhibited on the right side in the source code view. After clicking on the orange item “IDP.6” (marked as “2”) which stands for Illegal Dereference of Pointer, the source code view is updated to indicate the location of this orange warning. Moreover, after clicking on the orange section with the grey background of the code that is marked as “3” in Fig. 5-12, a warning message window is opened as shown in Fig. 5-13. This warning message window points out that the pointer in the source code line 1463, column 7 may be outside its bounds.

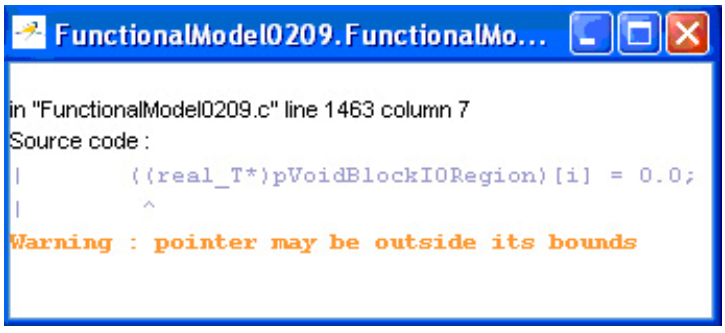


Fig. 5-13. Warning message window of the C code produced from RTW.

Furthermore, after the analysis is performed from PolySpace, textual files are produced which can be found in Appendix D and Appendix E for the analysis results of the C code produced from ARTiSAN Studio and RTW EC respectively. They can be used to create Excel reports. The report contains several spreadsheets related to the application analyzed.

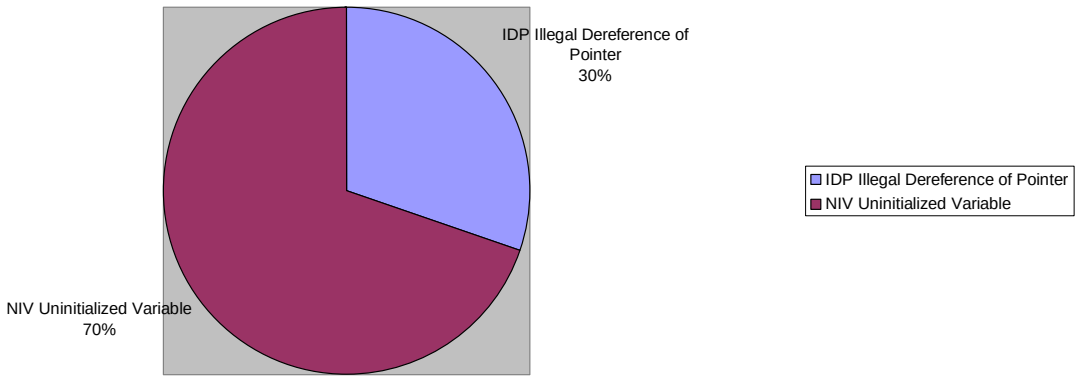


Fig. 5-14. Distribution of orange checks by categories of the C code produced from state machine diagram.

Fig. 5-14 shows the “Orange Check Distribution” spreadsheet which is used to present the distribution of orange checks by categories of the C code produced from eight state machine diagrams of the model built in ARTiSAN Studio. The orange warnings consist of two types of software defects, i.e. “Uninitialized Variable” and “Illegal Dereference of Pointer”. The uninitialized variable takes up 70% of the orange warnings and illegal dereference of pointer forms the remaining 30%.

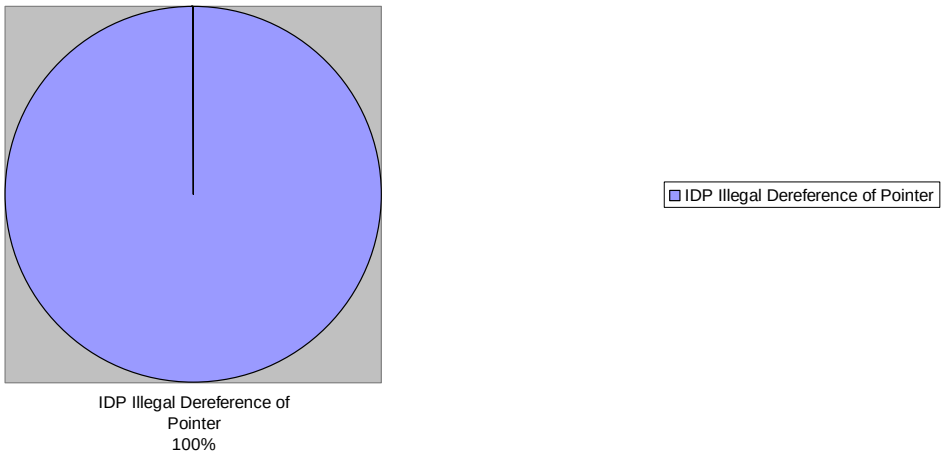


Fig. 5-15. Distribution of orange checks by categories of the C code produced from Stateflow diagram.

By contrast with Fig. 5-14, Fig. 5-15 shows the “Orange Check Distribution” spreadsheet that is used to explain the distribution of orange checks by categories of the C code produced from nine Stateflow diagrams in the Simulink/Stateflow model. The orange warnings are made up by “Illegal Dereference of Pointer” only.

Fig. 5-16 presents the “Distribution of checks by file” spreadsheet of the C code produced from eight state machine diagrams of the model built in ARTiSAN Studio. In this diagram, the X-axis indicates the number of checks. The Y-axis lists the files which have been analyzed or used during the analysis. The C file “_polyspace_main.c” is the main procedure which is automatically generated by PolySpace to carry out static analysis. Thus, this C code is error free in about 720 operations which is shown as a flat green bar. The “Driver_Information_System.c” is produced by the ARTiSAN Studio C Synchronizer. According to the scale on the X-axis, this code has nearly 400 orange warnings and almost 300 unreachable operations in the code.

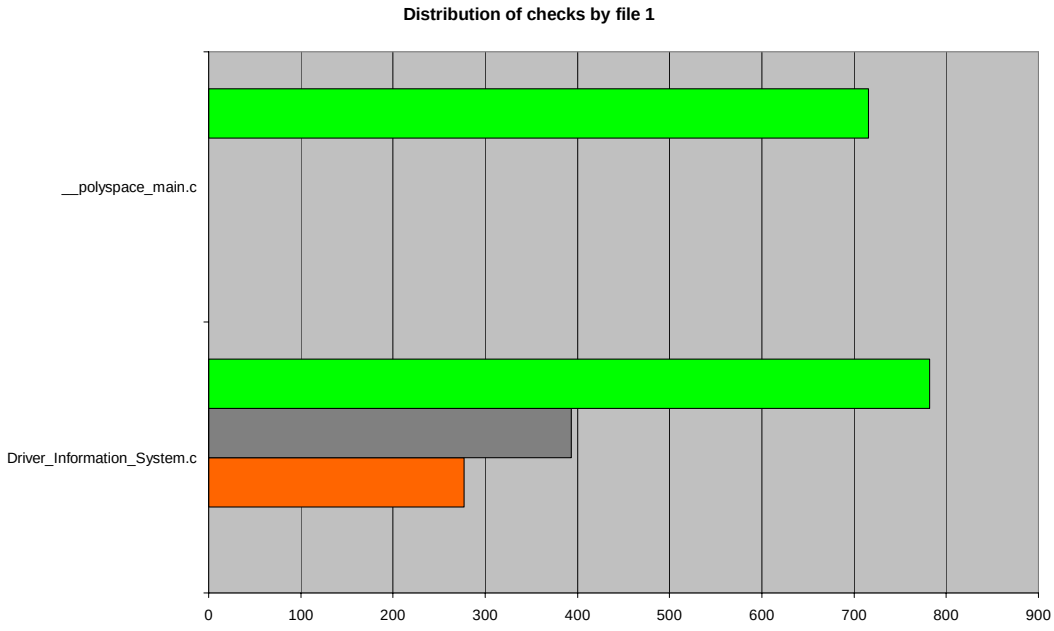


Fig. 5-16. Distribution of checks by file of the C code produced from state machine diagrams of the model built in ARTiSAN Studio.

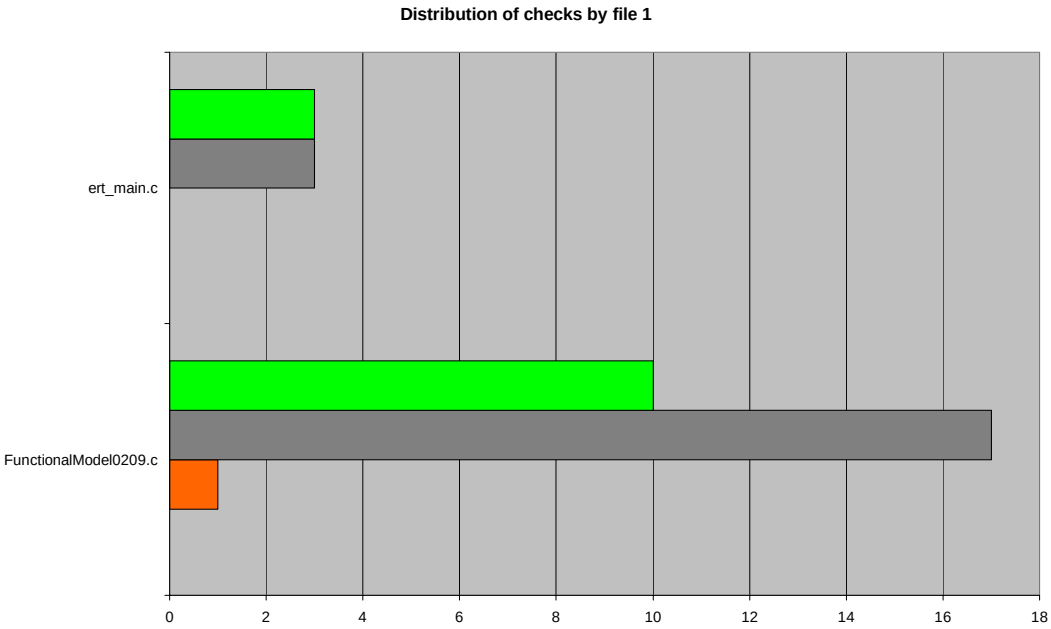


Fig. 5-17. Distribution of checks by file of the C code produced from Stateflow diagrams in Simulink/Stateflow model.

Fig. 5-17 displays the “Distribution of checks by file” spreadsheet of the C code produced from nine Stateflow diagrams in the Simulink/Stateflow model. Similar to Fig. 5-16, the X-axis indicates the number of checks. The Y-axis illustrates the “ert_main.c” and the “FunctionalModel0209.c”. Both C codes are generated by RTW EC and have been analyzed by PolySpace. “ert_main.c” contains three safe operations and three unreachable operations which are shown in the green and grey bars in Fig. 5-17 respectively. In “FunctionalModel0209.c”, there are 10 safe operations, 17 unreachable operations and only one orange warning in the code. In comparison with Fig. 5-16, it is clear that PolySpace performed significantly fewer checks in total for these C files. This is because the RTW EC produced a short C code in terms of length.

The above spreadsheets facilitate an overview of the automatically generated C code for the software engineer. It enables engineers to seek improvement in code

generation such as developing an auto coding platform for consistent, robust and reliable code delivery.

1	RTE Statistics						
2	**Check category**	**Check detail**	**R**	**O**	**Gy**	**Gr**	**% proved**
3	OBAI	Out of Bounds Array Index	0	0	0	38	100.00%
4	NIVL	Uninitialized Local Variable	0	0	0	215	100.00%
5	IDP	Illegal Dereference of Pointer	0	84	131	205	80.00%
6	NIP	Uninitialized Pointer	0	0	131	537	100.00%
7	NIV	Uninitialized Variable	0	193	131	100	54.48%
8	IRV	Initialized Value Returned	0	0	0	23	100.00%
9	COR	Other Correctness Conditions	0	0	0	152	100.00%
10	ASRT	User Assertion Failure	0	0	0	0	N/A
11	POW	Power Must Be Positive	0	0	0	0	N/A
12	ZDV	Division by Zero	0	0	0	152	100.00%
13	SHF	Shift Amount Within Bounds	0	0	0	0	N/A
14	OVFL	Overflow	0	0	0	38	100.00%
15	UNFL	Underflow	0	0	0	38	100.00%
16	UOVFL	Underflow or Overflow	0	0	0	0	N/A
17	EXCP	Arithmetic Exceptions	0	0	0	0	N/A
18	NTC	Non Termination of Call	0	0	0	0	N/A
19	k-NTC	Known Non Termination of Call	0	0	0	0	N/A
20	NTL	Non Termination of Loop	0	0	0	0	N/A
21	UNR	Unreachable Code	0	0	0	0	N/A
22	UNP	Uncalled Procedure	0	0	0	0	N/A
23	IPT	Inspection Point	0	0	0	0	N/A
24	OTH	other checks	0	0	0	0	N/A
25	EXC	Exception handling	0	0	0	0	N/A
26	OOP	Object Oriented Programming	0	0	0	0	N/A
27	CPP	C++	0	0	0	0	N/A
28	NNR	Non Null Receiver	0	0	0	0	N/A
29	FRV	Function Returns a Value	0	0	0	0	N/A
30	INF	Informative check	0	0	0	0	N/A
31	Total :		0	277	393	1498	87.22%

Fig. 5-18. RTE view of the C code produced from state machine diagrams of the model built in ARTiSAN Studio.

As shown in Fig. 5-18, the Run Time Error (RTE) view is presented in the “Check Synthesis” spreadsheet. It contains all statistics about checks and colours in a summary table. Fig. 5-18 represents the RTE view of the C code produced from eight state machine diagrams in SysML model. In this table, it can be seen that ten

types of errors have been checked. They are pointed out in lines 3, 4, 5, 6, 7, 8, 9, 12, 14 and 15 respectively. Based on Fig. 5-18, there are no certain errors which occur on each execution. There are 1,498 safe operations, 393 unreachable operations and 277 orange warnings in the code. This code is 87.22% proven overall.

1	RTE Statistics						
2	Check category	Check detail	R	O	Gy	Gr	% proved
3	OBAI	Out of Bounds Array Index	0	0	1	0	100.00%
4	NIVL	Uninitialized Local Variable	0	0	7	3	100.00%
5	IDP	Illegal Dereference of Pointer	0	1	1	2	75.00%
6	NIP	Uninitialized Pointer	0	0	2	4	100.00%
7	NIV	Uninitialized Variable	0	0	0	0	N/A
8	IRV	Initialized Value Returned	0	0	0	0	N/A
9	COR	Other Correctness Conditions	0	0	0	0	N/A
10	ASRT	User Assertion Failure	0	0	0	0	N/A
11	POW	Power Must Be Positive	0	0	0	0	N/A
12	ZDV	Division by Zero	0	0	1	0	100.00%
13	SHF	Shift Amount Within Bounds	0	0	0	0	N/A
14	OVFL	Overflow	0	0	4	2	100.00%
15	UNFL	Underflow	0	0	4	2	100.00%
16	UOVFL	Underflow or Overflow	0	0	0	0	N/A
17	EXCP	Arithmetic Exceptions	0	0	0	0	N/A
18	NTC	Non Termination of Call	0	0	0	0	N/A
19	k-NTC	Known Non Termination of Call	0	0	0	0	N/A
20	NTL	Non Termination of Loop	0	0	0	0	N/A
21	UNR	Unreachable Code	0	0	0	0	N/A
22	UNP	Uncalled Procedure	0	0	0	0	N/A
23	IPT	Inspection Point	0	0	0	0	N/A
24	OTH	other checks	0	0	0	0	N/A
25	EXC	Exception handling	0	0	0	0	N/A
26	OOP	Object Oriented Programming	0	0	0	0	N/A
27	CPP	C++	0	0	0	0	N/A
28	NNR	Non Null Receiver	0	0	0	0	N/A
29	FRV	Function Returns a Value	0	0	0	0	N/A
30	INF	Informative check	0	0	0	0	N/A
31	Total :		0	1	20	13	97.06%

Fig. 5-19. RTE view of the C code produced from Stateflow diagrams in Simulink/Stateflow model.

By contrast with Fig. 5-18, Fig. 5-19 shows the RTE view of the C code generated from nine Stateflow diagrams of the Simulink/Stateflow model. The result is summarised at the bottom of the table. No definite run-time errors are found in this code. There are 13 safe operations, 20 unreachable operations and there is only one orange report which is a possible error. This code is 97.06% verified overall.

From the comparison above, there are three more error types that could be applied to the code produced by ARTiSAN Studio from state machine diagrams which are listed in lines 7, 8 and 9 respectively. “Uninitialized Variable” in line 7 which is an error type possibly applies to this C code produced the most orange warnings that cause this code to have a lower proven rate.

5.4 Code inspection and analysis

Section 5.3 shows a significant difference between the number of operations of the C code produced by ARTiSAN Studio and RTW EC. This section carries out further investigation based on the results observed from the static analysis of the software code.

The C code generation for SysML model is performed in ARTiSAN Studio version 6.1. There is no option available in the C synchronizer to optimize the code generation process. As described in Section 5.2, the default options are selected when RTW EC produced the C code for the Simulink/Stateflow model. The fist experiment is to explore how the length of the C code changes when different target files are selected.

Table 5-2. C code length under different target files.

Target file	Description	Length of the C code (lines)
ert.tlc	Real-Time Workshop Embedded Coder with no auto configuration.	1,818
ert_shrlib.tlc	Real-Time Workshop Embedded Coder with host-based shared library target.	1,247
grt.tlc	Generic real-time target	1,474
rtwin.tlc	Real-time Windows target	1,457
rtwsfcn.tlc	S-function target	1,887

As shown in the Table 5-2, five target files are selected for the C code generation. The C code can be produced for different purposes by selecting the target file which is described in the middle column. The right column specifies the length of the C code generated. Comparing with the 4,311 lines C code produced from the functionally equivalent model in ARTiSAN Studio, the significant difference in length still exists. Additional static analysis is carried out for the code produced by selecting “ert_shrlib.tlc” target file.

1	RTE Statistics						
2	**Check category**	**Check detail**	**R**	**O**	**Gy**	**Gr**	**% proved**
3	OBAI	Out of Bounds Array Index	0	0	0	1	100.00%
4	NIVL	Uninitialized Local Variable	0	0	2	5	100.00%
5	IDP	Illegal Dereference of Pointer	0	0	0	2	100.00%
6	NIP	Uninitialized Pointer	0	0	0	3	100.00%
7	NIV	Uninitialized Variable	0	0	0	0	N/A
8	IRV	Initialized Value Returned	0	0	0	0	N/A
9	COR	Other Correctness Conditions	0	0	0	0	N/A
10	ASRT	User Assertion Failure	0	0	0	0	N/A
11	POW	Power Must Be Positive	0	0	0	0	N/A
12	ZDV	Division by Zero	0	0	1	0	100.00%
13	SHF	Shift Amount Within Bounds	0	0	0	0	N/A
14	OVFL	Overflow	0	0	2	2	100.00%
15	UNFL	Underflow	0	0	2	2	100.00%
16	UOVFL	Underflow or Overflow	0	0	0	0	N/A
17	EXCP	Arithmetic Exceptions	0	0	0	0	N/A
18	NTC	Non Termination of Call	0	0	0	0	N/A
19	k-NTC	Known Non Termination of Call	0	0	0	0	N/A
20	NTL	Non Termination of Loop	0	0	0	0	N/A
21	UNR	Unreachable Code	0	0	0	0	N/A
22	UNP	Uncalled Procedure	0	0	0	0	N/A
23	IPT	Inspection Point	0	0	0	0	N/A
24	OTH	other checks	0	0	0	0	N/A
25	EXC	Exception handling	0	0	0	0	N/A
26	OOP	Object Oriented Programming	0	0	0	0	N/A
27	CPP	C++	0	0	0	0	N/A
28	NNR	Non Null Receiver	0	0	0	0	N/A
29	FRV	Function Returns a Value	0	0	0	0	N/A
30	INF	Informative check	0	0	0	0	N/A
31	Total :		0	0	7	15	100.00%

Fig. 5-20. RTE view of the C code produced by selecting “ert_shrlib.tlc” target file from Stateflow diagrams in Simulink/Stateflow model.

Fig. 5-20 shows the RTE view of the C code generated by selecting the “ert_shrlib.tlc” target file from Stateflow diagrams of the Simulink/Stateflow model. The result is summarised at the bottom of the table. No definite run-time errors are found in this code. There are 15 safe operations, 7 unreachable operations and there is no orange warning in the code. This code is 100.00% proven overall.

The next experiment focuses on finding out the impact of selecting different solvers. The experiment is based on “Code 1” in Fig. 5-6 where the default settings are employed as explained in Section 5.2. The default setting for the solver is “ode3” which is shown in Fig. 5-8. As explained in section 5.3, this C code has 1,818 lines and there are 13 safe operations, 20 unreachable operations and there is one orange warning. This code is 97.06% verified overall. Three different solvers are selected for the code generation, namely, “ode1”, “ode5” and “ode14x”. The static analysis shows the C code produced with different solvers have same length and operations as shown in RTE view of the Fig. 5-19. There is no distinct difference highlighted among these C codes.

To investigate the reason for the huge difference between the numbers of operations of the C code produced in ARTiSAN Studio and in Simulink/Stateflow, the “Code 5” in Fig. 5-5 and the “Code 1” in Fig. 5-6 are manually inspected and compared. They are detailed as follows.

There are two types of files in the C program produced. The header file, with an extension name of '.h', defines all the variables and functions using the program; meanwhile, the source file, with an extension name of '.c', includes all the operation that the C code performs.

Generally speaking, the readability of “Code 1” is much higher than “Code 5”. In particular, there are three major variables and three functions in “Code 1”. They are defined in FunctionalModel0209.h line 379 to line 391.

```
/* Block parameters (auto storage) */
extern Parameters_FunctionalModel0209 FunctionalModel0209_P;

/* Block signals (auto storage) */
```

```
extern BlockIO_FunctionalModel0209 FunctionalModel0209_B;

/* Block states (auto storage) */
extern D_Work_FunctionalModel0209 FunctionalModel0209_DWork;

/* Model entry point functions */
extern void FunctionalModel0209_initialize(void);
extern void FunctionalModel0209_step(void);
extern void FunctionalModel0209_terminate(void);
```

Most of the functions in the “Code 1” are realized by above three variables and three functions.

Variable “D_Work_FunctionalModel0209” records the states of all the state blocks in the “Code 1”.

Variable “BlockIO_FunctionalModel0209” records all the signals that modify the state blocks.

Variable “Parameters_FunctionalModel0209” records all the parameters in the state blocks.

Function “FunctionalModel0209_initialize” initializes the state blocks and the entire state chart.

Function “FunctionalModel0209_step” modifies the state blocks in the state chart.

Function “FunctionalModel0209_terminate” eliminates all the blocks in the memory.

All the operations in the “Code 1” can be summarized as follows. The program firstly uses function “FunctionalModel0209_initialize” to initialize the blocks and save the information in the blocks in “D_Work_FunctionalModel0209”, then uses

function “FunctionalModel0209_step” to generate the signal , i.e. dataflow, saves it in “BlockIO_FunctionalModel0209” and uses the signal to modify the parameters for the blocks, which are recorded in the variable “Parameters_FunctionalModel0209”. Finally, when the user needs to exit the program, all the variables and blocks are eliminated by the function “FunctionalModel0209_terminate”.

From the above explanation, it can be concluded that all of the components, such as the “System_States”, “Screen_States”, etc, in the C code are encapsulated in the three variables, while in the processing, they cannot be seen by the inspector, e.g. the PolySpace static analysis tool. Meanwhile, all the operations, such as “power off the screen”, “turn on the engine”, i.e. “Screen_off”, “Ignition_On” in the “Code 1” are encapsulated in the three functions, and are also not seen by the inspector.

By contrast with “Code 1”, “Code 5” defines every component functions and variables separately. For example, “RtsDriver_Information_System_States” defines the states of the system separately in lines 15 to 24 of “Driver_Information_System.h” as shown below.

```
/*
\ART_SMG :: Created for state : Driver_Information_System
*/
enum RtsDriver_Information_System_States
{
    Screen_Power_Off,
    Initial_Screen_Showing_Company_Logo,
    Display,
    NotIn_Driver_Information_System
```

```
};
```

“RtsDriver_Information_SystemDisplay_States” defines the states of the display separately in lines 26 to 39 of “Driver_Information_System.h” as shown below.

```
/*
\ART_SMG :: Created for state : Display
*/
enum RtsDriver_Information_SystemDisplay_States
{
    Entertainment,
    Home,
    Navigation,
    OffRoad_Information,
    OnRoad_Information,
    Phone,
    Settings,
    NotIn_Display
};
```

In contrast, these variables are included in the variable “D_Work_FunctionalModel0209” in “Code 1”.

In addition, in lines 394 to 402 of the “Driver_Information_System.h”, two operations are defined which are “Ignition_On” and “4X4_Info_Soft_Key_Pressed” as below,

```
/*
\ART_SMG :: Created for state : Screen_Power_Off
*/
void RtsIgnition_On(struct Driver_Information_System* this);
```

```
 /*
\ART_SMG :: Created for state : Display
*/
void Rts4X4_Info_Soft_Key_Pressed(struct Driver_Information_System* this);
```

In contrast, in “Code 1”, it is part of the function “FunctionalModel0209_step” which is not defined separately.

To summarize, it is obvious that there are more definitions in the header file of “Code 1”. The header file in “Code 1” has 756 lines and is 34kb in size whereas the header file of “Code 5” has 576 lines and is 14kb in size. Generally speaking, the well-defined header file, which represents a well-designed data structure, will increase the efficiency of the program and thus results in shorter C code. Consequently, “Code 1” has 78Kb 1,818 lines whereas “Code 5” has 4,311 lines and is 113kb in size. Moreover, from the human-inspector's point of view, “Code 1” is more compact, more advanced and has a higher readability. Analyzed by the PolySpace, “Code 1” encapsulates most component variables and functions into the aforementioned three variables and functions and consequently uses fewer variables and fewer functions in the code. Therefore, PolySpace only produces 34 tests as detailed in Fig. 5-19. In contrast, PolySpace has to test all the variables and functions in “Code 5” which is 2,168 in total as shown in Fig. 5-18. The potential reason is that the Simulink is a component program of MATLAB. It is well-known that MATLAB is designed based on JAVA, which is more advanced than C. Therefore, the C code produced by Simulink is more advanced with encapsulation than the C code generated by ARTiSAN Studio which has to use a larger number of pointers to handle the huge amount of functions and variables indexed at lines 5, 6 and 7 in Fig. 5-18.

5.5 Discussion

Traditional automotive embedded software development involves paper designs and hand coding, followed by verification activities such as code inspections. These activities include manual interaction and lack of tool automation, and as a consequence they are error prone and time consuming. Modelling tools such as ARTiSAN Studio and MATLAB/Simulink not only provide insight into the dynamic behaviour of the system by enabling the simulation of the function model as described in Section 5.1, but also enable automatic code generation for embedded software delivery as presented in Section 5.2.

The development of ECUs with embedded software plays an important role to address the challenges of increasing complexity while developing and maintaining electronic applications within the automotive system. There are several advantages of utilizing the software tools such as C Synchronizer and RTW EC within the model-based design, e.g., software designed in the model can be executed and verified in simulation, auto coding from the model reduces the possibility of the error being introduced at this phase of development. It is especially crucial for achieving advanced functions in the automotive SoS, that the correctness of the model and software code directly affect the quality of the embedded software in the ECU and, therefore, the overall SoS. The automatic code generation and static analysis are highly important benefits that make the development of an automotive electronic SoS efficient and effective.

The static analysis tool PolySpace can analyze C code and relate potential defects found therein back to the design model from which the code was generated. Hence, defects can be avoided and the quality of the software code can be ensured. This chapter considers the auto coding capabilities of the C Synchronizer in

ARTiSAN Studio and RTW EC in MATLAB/Simulink. The C code generated from ARTiSAN Studio for state machine diagrams is twice as long as that produced by Simulink/Statelow. Within an embedded software environment, longer code could lead to complexity of software. Complex software is difficult to understand and maintain. In addition, longer code requires larger memory on the ECU to execute, which means additional costs to the manufacturers.

The results show that the Simulink/Stateflow is more capable of producing efficient and error free C code for the software delivery. But ARTiSAN Studio enables the engineer to produce the code from a certain component of the model such as state machine diagrams to an entire model including both a structure model and a function model. Hence, the code generated from the SysML model in ARTiSAN Studio can provide a more comprehensive coverage of the whole system [117].

In summary, to develop an automotive electronic SoS, building the structure and function model in SysML by using ARTiSAN Studio is an essential and effective way to gain a comprehensive understanding of the entire SoS. In dealing with software delivery, developing the function model in Simulink/Stateflow and automatically generating the code from such a model is the preferred and more efficient approach which is the conclusion of this chapter.

Chapter 6

Real-time simulation and animation of the 4×4 Information System interface

This chapter proposes a novel approach to verify advanced functions of automotive electronic systems. It utilises the function model built in Simulink/Stateflow and the C code produced from this model through RTW for the real-time platform target, i.e. the dSPACE Simulator for implementing the real-time simulation and animation of the 4×4 Information System interface. The feasibility is proved and the research approach taken will be demonstrated in this chapter.

6.1 Introduction

Application of the offline simulation of state machine diagrams in the function model built in ARTiSAN Studio and Stateflow diagrams in Simulink/Stateflow model is demonstrated in Chapter 5. In addition, ARTiSAN Studio facilitates the animation of the sequence diagrams in the model through the object animator. Fig. 6-1 shows the animation of “Sequence diagram 3: view steering angle (high level).” that is presented and explained in Fig. 4-7 of Chapter 4. The actor is

represented at the top of this diagram. Three grey circles stand for the interface devices. The “Driver Information System Application Software” is displayed in the square at the bottom of the diagram. When the animation starts, black lines link objects together in a time sequence following the direction of data transfer between objects that are defined in the sequence diagram. The animation can be paused and restarted to test all possible conditions.

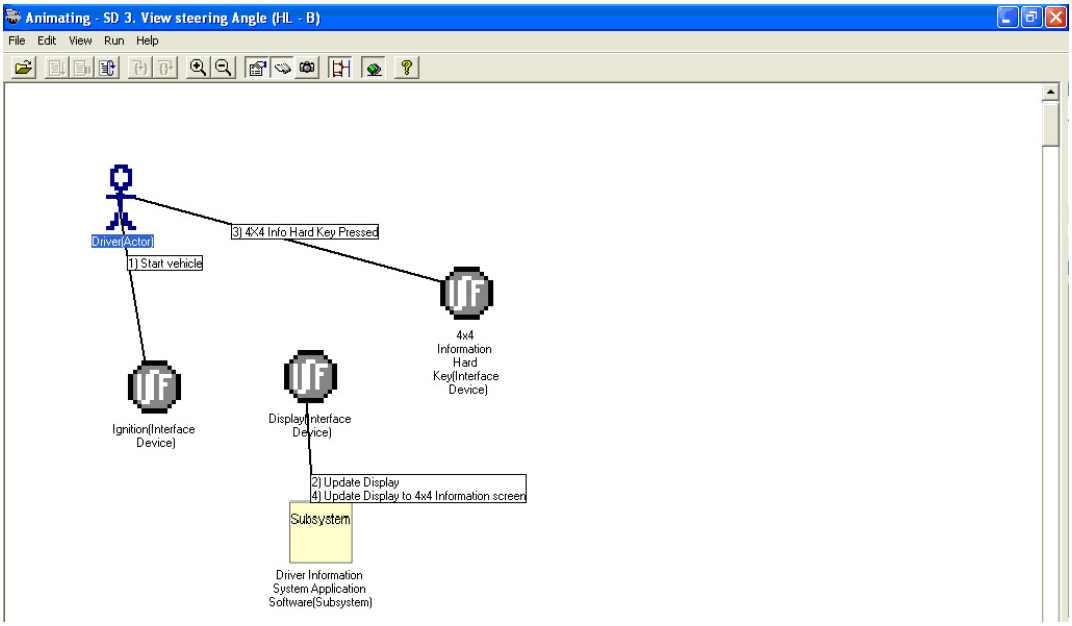


Fig. 6-1. Animation of sequence diagram in ARTiSAN Studio.

However, the simulation of the Stateflow diagrams in Simulink/Stateflow, the simulation of state machine diagrams and animation of the sequence diagrams in ARTiSAN Studio are all offline simulations. Real-time simulation and testing are required for robust system design to deliver flawless software for the automotive electronic system.

6.2 Experimental set-up

dSPACE ControlDesk is an advanced tool to manage real-time experiments in the process of function development. The developer can build virtual instrument panels

and have complete control over Simulink and real-time simulations. Typical application in the development of the automotive electronic system is used to simulate driving cycles and data acquisition [124]. This section demonstrates how the dSPACE ControlDesk is used to carry out real-time simulation of the Simulnk/Stateflow model. It will also explore the design, development and testing of the function of the 4×4 Information System from a novel approach.

6.2.1 Hardware platform

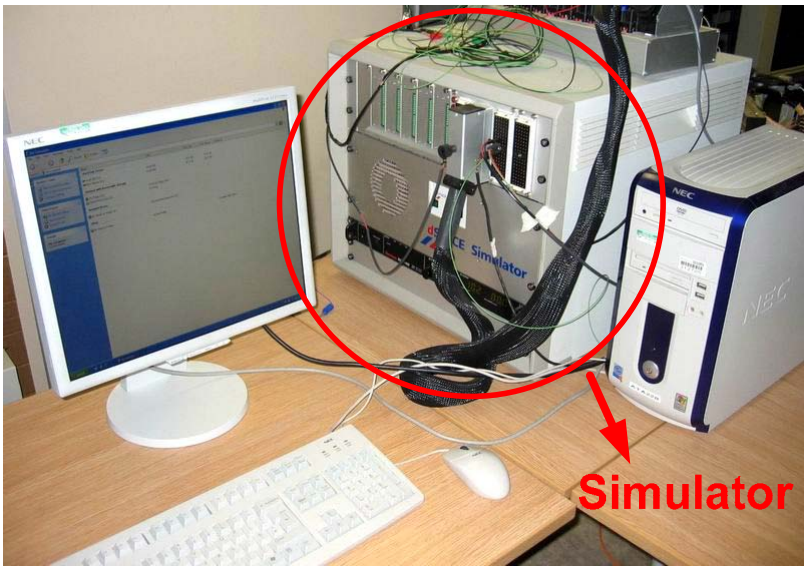


Fig. 6-2. Hardware platform for the real-time simulation.

The experiment in this chapter is based on the dSPACE Mid-Size Simulator [125]. As shown in Fig. 6-2, this simulator can be applied for functional integration tests, release tests and ECU diagnostics tests. In particular, it is capable of real-time simulations. The DS1006 processor board is used in this simulator as the processor board for very complex, large, and processing-intensive models. The board is built around the AMD Opteron™, a 64-bit server processor with 1MB L2 cache based on AMD64. The DS1006 also has 256 MB local memory for executing real-time models, 128 MB global memory for exchanging data with the host PC.

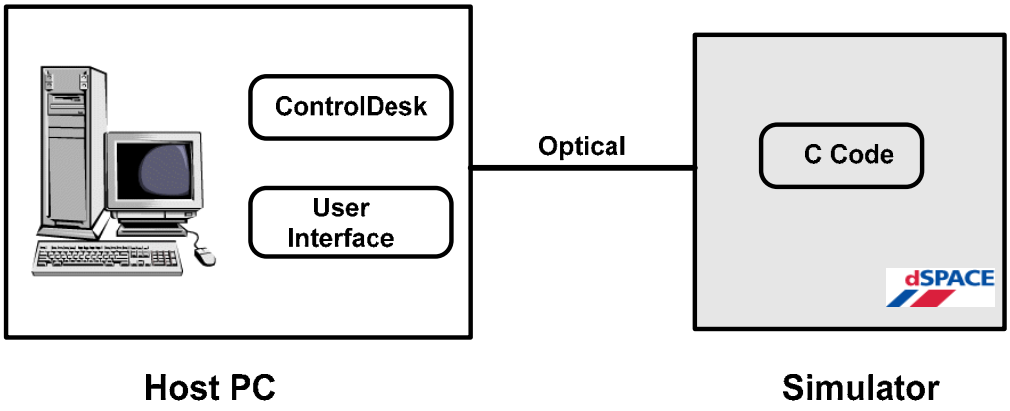


Fig. 6-3. Real-time simulation set-up.

Fig. 6-3 illustrates the set-up of this real-time simulation. The dSPACE ControlDesk is installed on the host PC. The model built in ControlDesk provides the user interface that allows the developer to control the simulation. The executable C code which is produced from the Simulink/Stateflow model by RTW is loaded and run in the dSPACE Simulator. The connection between the host PC and the Simulator is enabled by the laser optical cable that is capable of transferring large amounts of data during the real-time simulation.

6.2.2 Further development of the Simulink/Stateflow model

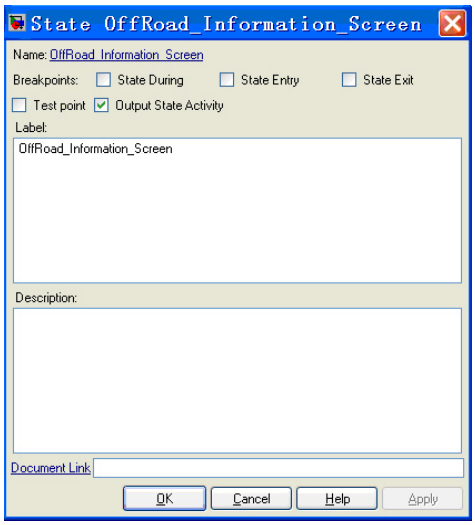


Fig. 6-4. Output state activities from states in the Simulink/Stateflow model.

The ControlDesk is used to build instrumentation panels for controlling and monitoring the variables of simulation. Therefore, the initial step of implementing real-time test of this model is to output each state as a variable as shown in Fig. 6-4. In order to do this, the properties of each state in the Stateflow model need to be opened and the box of “Output State Activity” has to be ticked. Each output state creates a port on the right hand side of the state chart in the Simulink model as shown in Fig. 6-6. In Fig. 6-5, three states have been exported. They are “OffRoad_Information_Screen”, “CompassView” and “ChassisView”. The display is connected to these outputs to observe the output signals.

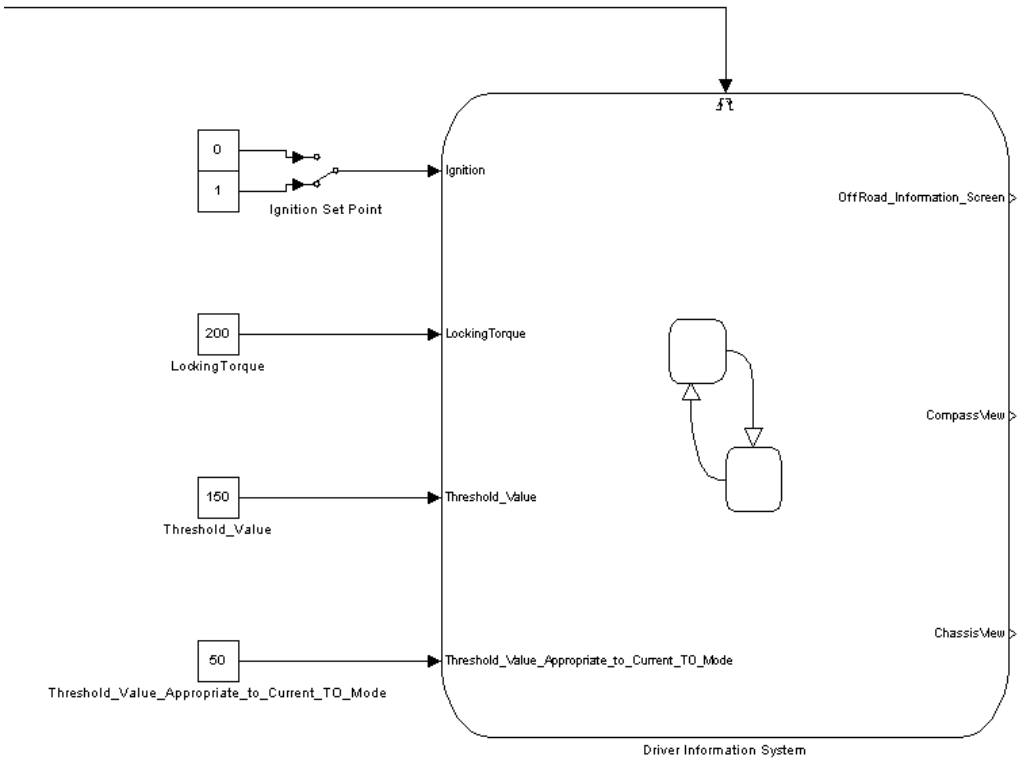


Fig. 6-5. Output state activities in Simulink.

As shown in Fig. 6-6, simulation is performed in order to verify output signals in the model. The display of “OffRoad_Information_Screen” and “ChassisView” showing “1” stands for active states. The number “0” in the display of “CompassView” indicates that it is an inactive state. All the states in the

Simulink/Stateflow model have been exported as variables in this experiment. The output signals in the Simulink/Stateflow model are verified by checking the consistency between active states that are indicated by numbers in displays and active states that are represented in blue during simulation.

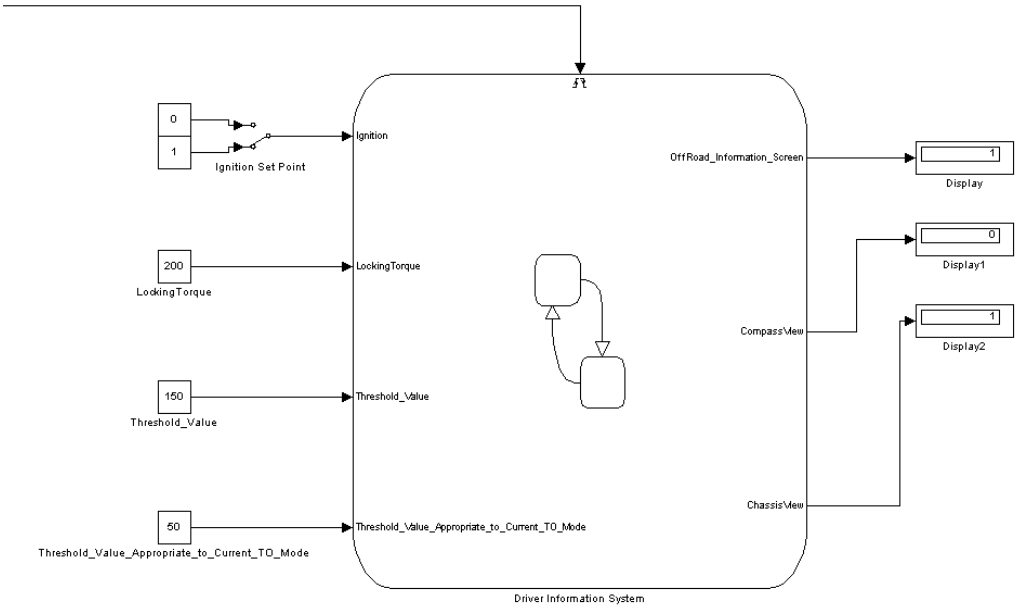


Fig. 6-6. Simulation of output state activities in Simulink.

Then customized C code is automatically generated by RTW from this Simulink/Stateflow model. In this experiment, the target file “rti1006.tlc” is selected to generate C code for the dSPACE DS1006 hardware platform.

6.3 Real-time simulation of the 4×4 Information System interface

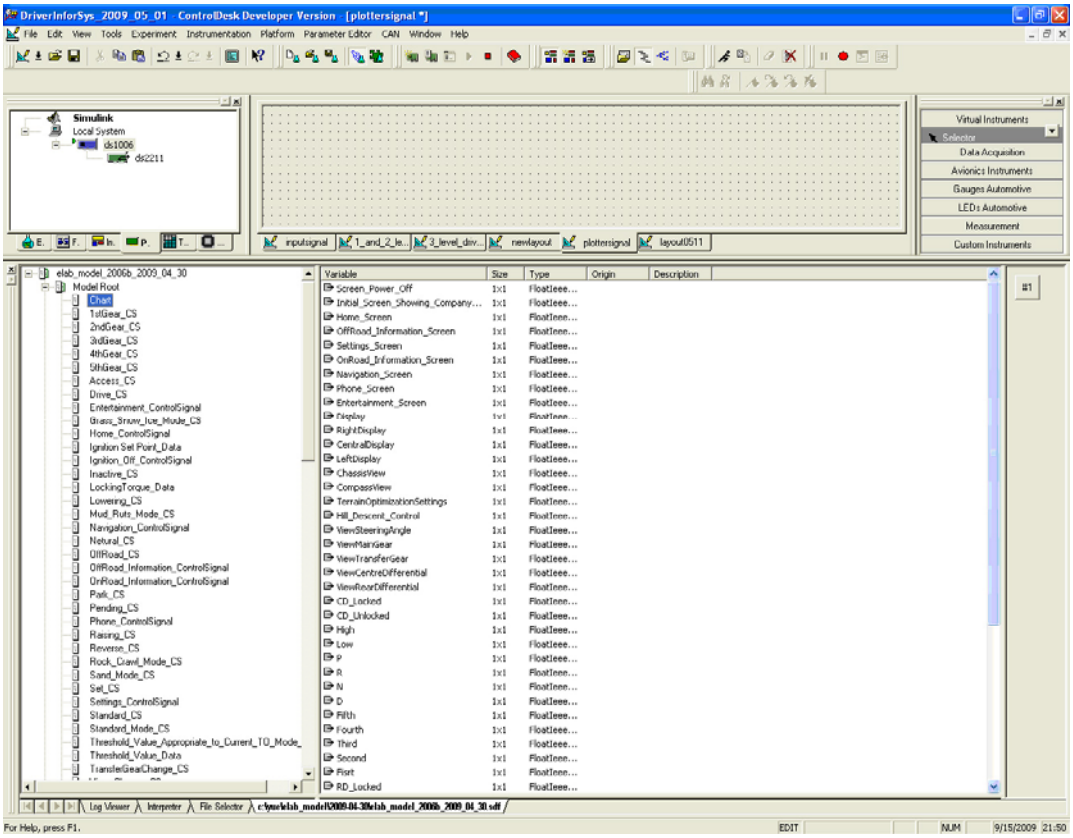


Fig. 6-7. Variables in ControlDesk model.

The generated C code contains all variables in the Simulink/Stateflow model and it can now be loaded into the dSPACE ControlDesk experiment. After the C code is loaded into the DS1006 board on the dSPACE simulator, the variables are listed at the bottom of the window as shown in Fig. 6-7. The ControlDesk contains a wide range of instruments that can be selected such as buttons, sliders, data acquisition instruments, etc., as shown in the right of Fig. 6-8. The variables will then be linked with selected instruments. To assign variables, they are dragged from the variable browser onto the instrument and these steps are repeated for all variables in this experiment. Selected instruments can now be put in operation by switching to animation mode. Fig. 6-9 shows part of the layout which contains control signals.

Four knobs at the top of this diagram are used to control the input value which is modelled on the left of the state chart in the Simulink/Stateflow model as illustrated in Fig. 6-5 and Fig. 6-6. Grey rectangles in this diagram are “OnOffButton”s. They are utilized to represent the constant blocks in the Simulink model to generate control signals. The “RadioButton”s on the right of this diagram are used to switch between “ChassisView” and “CompassView”, “Low” transfer gear and “High” transfer gear respectively. All the output states are linked with “MultiStateLED”. Fig. 6-10 shows one of the output displays in comparison with Fig. 5-2 after the simulation is enabled.

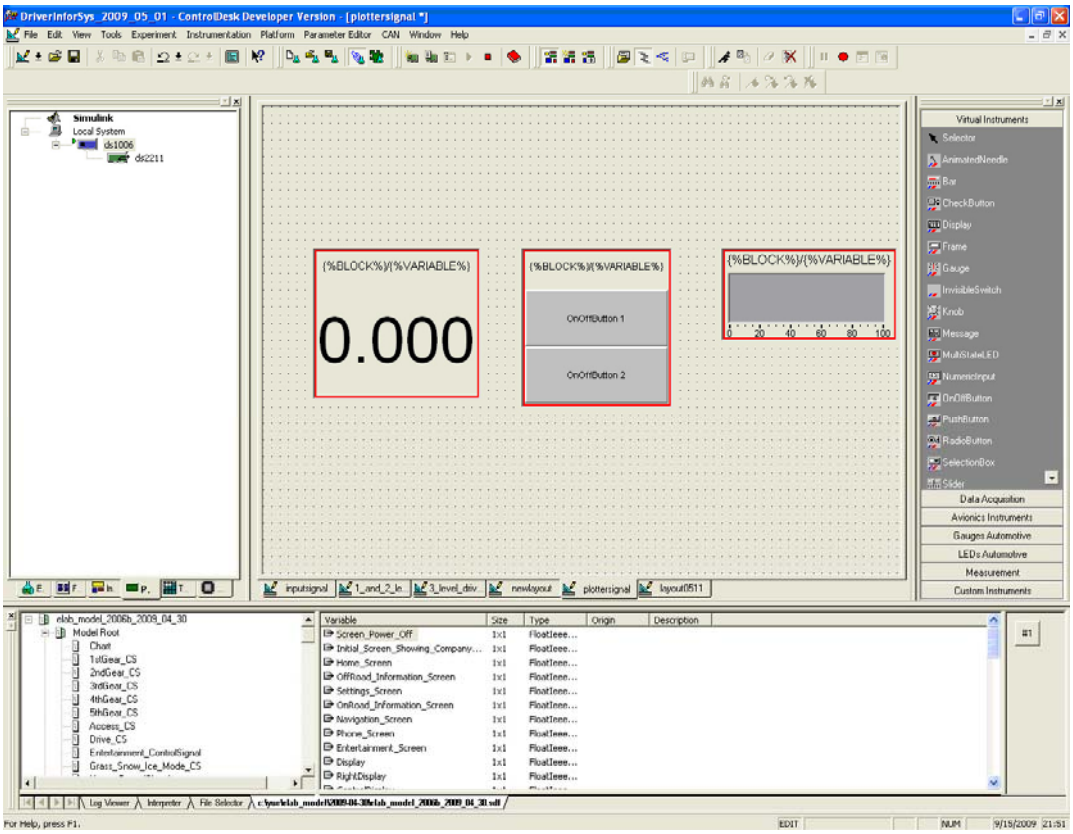


Fig. 6-8. Instruments in ControlDesk experiment.

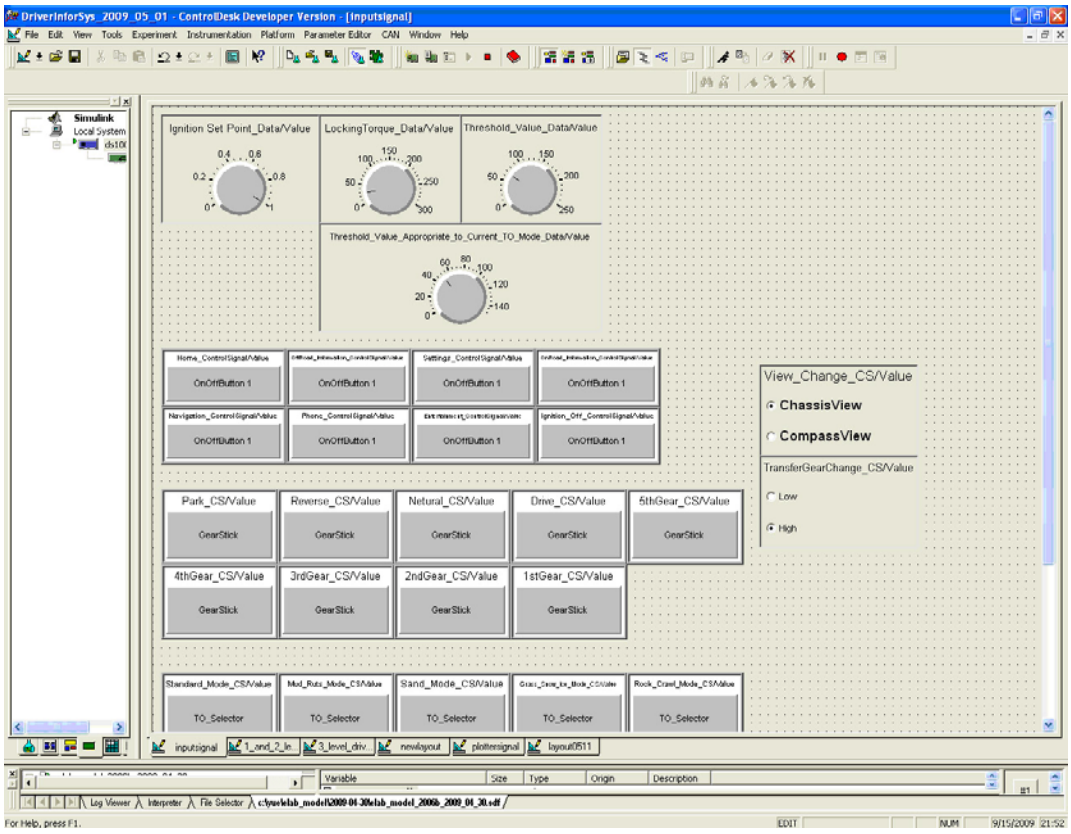


Fig. 6-9. Layout of input signals in ControlDesk.

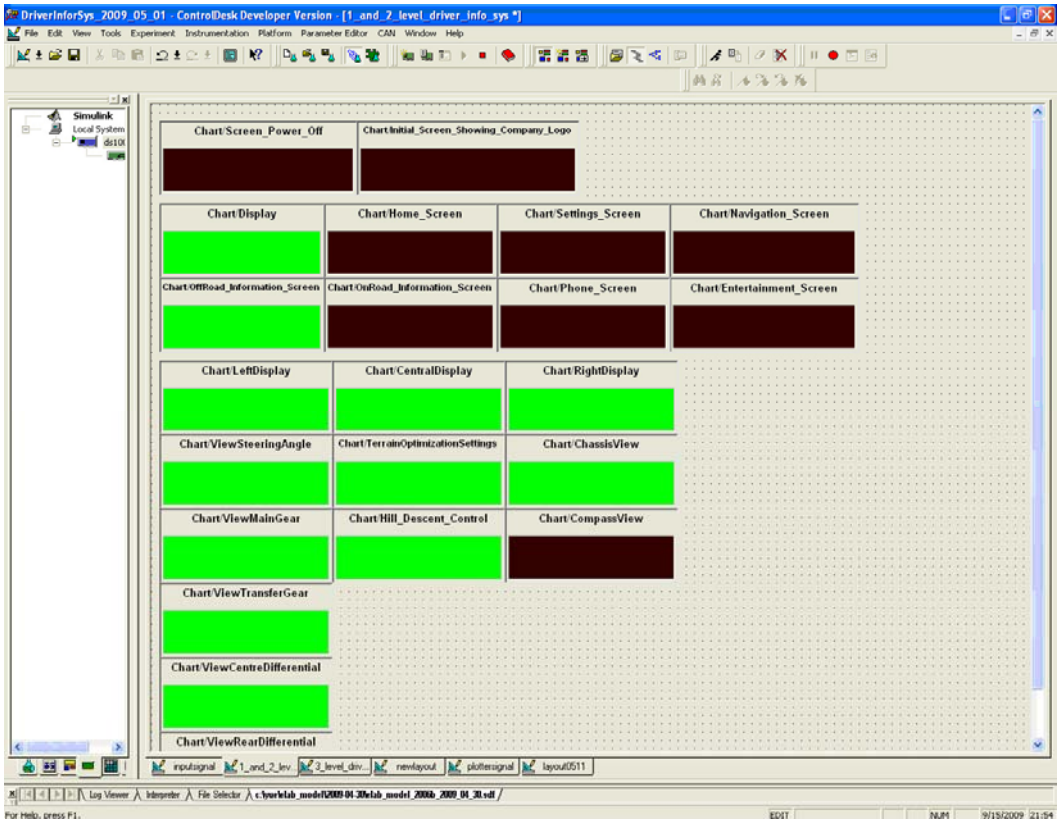


Fig. 6-10. Simulation of 4×4 information screen in ControlDesk.

As shown in Fig. 6-10, blocks turn into a green colour to represent active states when corresponding buttons in Fig. 6-9 are clicked. For instance, the screen currently shows 4×4 information. The information on the left and central area of the display is available to the driver. The right area is currently in chassis view. The functions of the 4×4 Information System are thereby simulated and tested on a real-time basis. However, there is a difference between the ControlDesk model and the actual 4×4 Information System displayed in terms of user interface. Therefore, animation of the real-time simulation is enabled to obtain a better visualization.

6.4 Animation of the real-time simulation

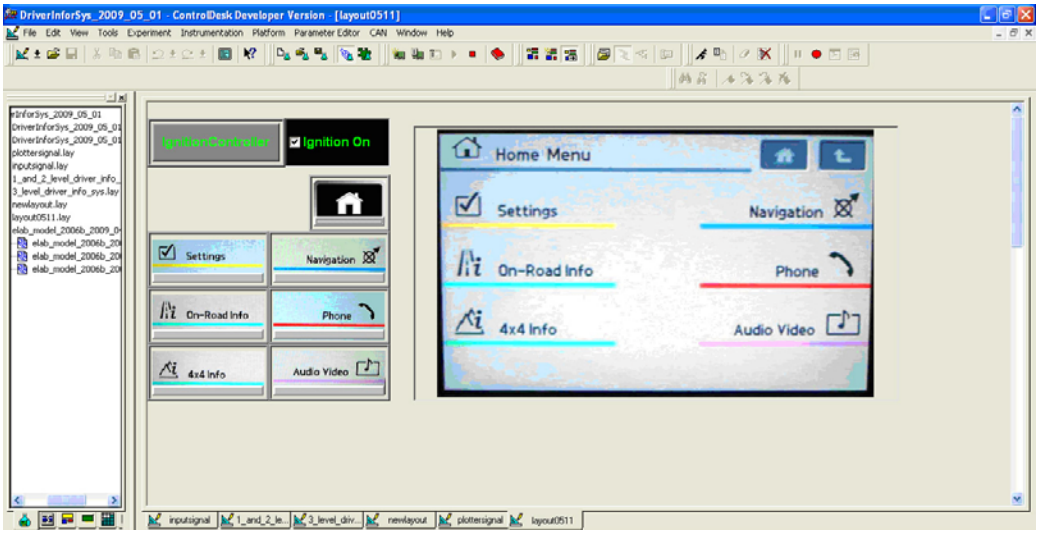


Fig. 6-11. Display shows “Home” screen during real-time animation.

The animation of the user interface of the 4×4 Information System is realized by customizing appointed instruments that benefits from comprehensive configuration options for instrument properties such as size, position, fonts and colours. For instance, the background picture of “MultiStateLED” which shows the status of the states is changed according to its role in the system. As shown in Fig. 6-11, the home screen of the Driver Information System is displayed on the right side of this

layout. The buttons for generating control signals are located on the left side of this layout. During the real-time animation, the developer clicks on the button, the display on the right side of the layout switching to corresponding pictures represents the mode changes. For example, when the developer clicks on the "Audio Video" button, the right area shows the "Radio" screen as displayed in Fig. 6-12. The "4×4 Info" screen appears when the "4×4 Info" button is clicked. The "4×4 Info" screen as shown in Fig. 6-13 represents the 4×4 Information System that is modelled in detail.

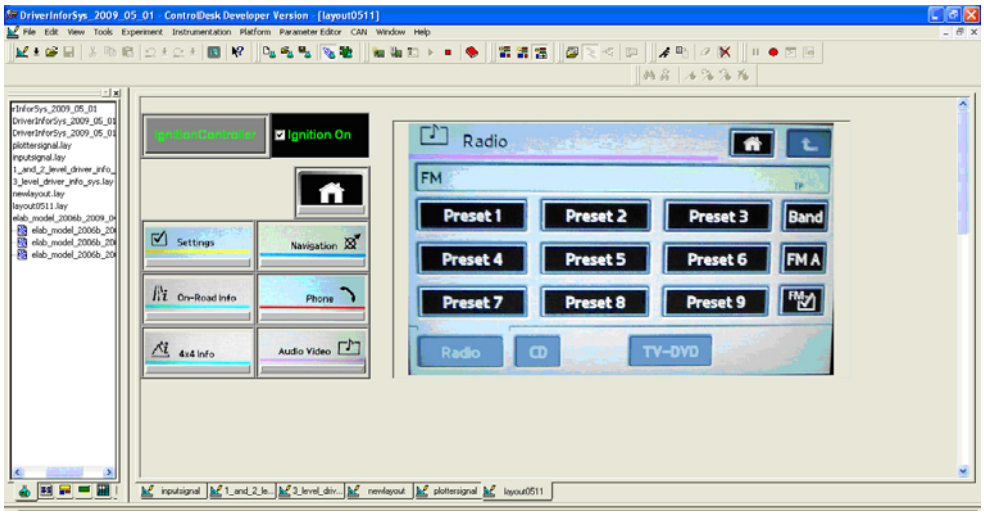


Fig. 6-12. Display shows "Audio Video" during real-time animation.

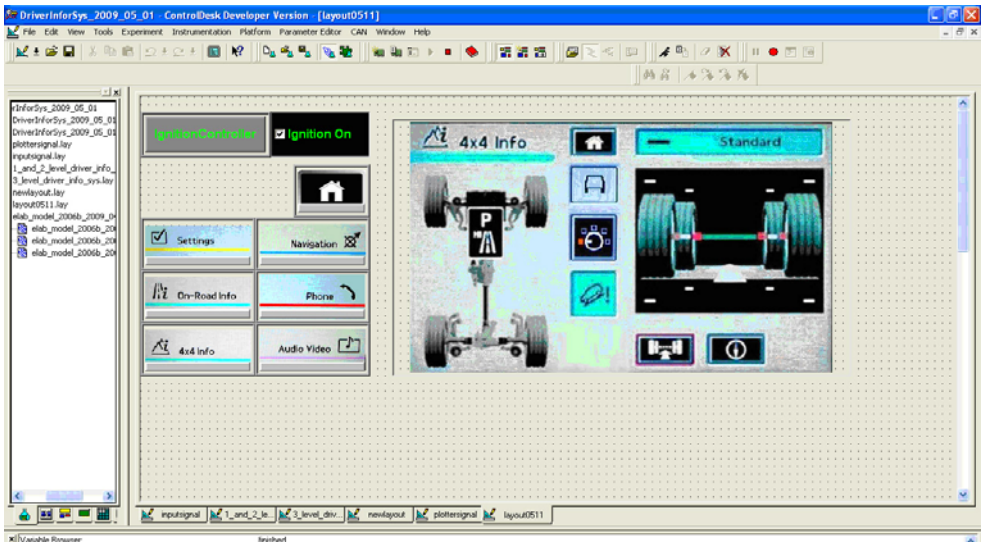


Fig. 6-13. Display shows "4×4 Info" during real-time animation.

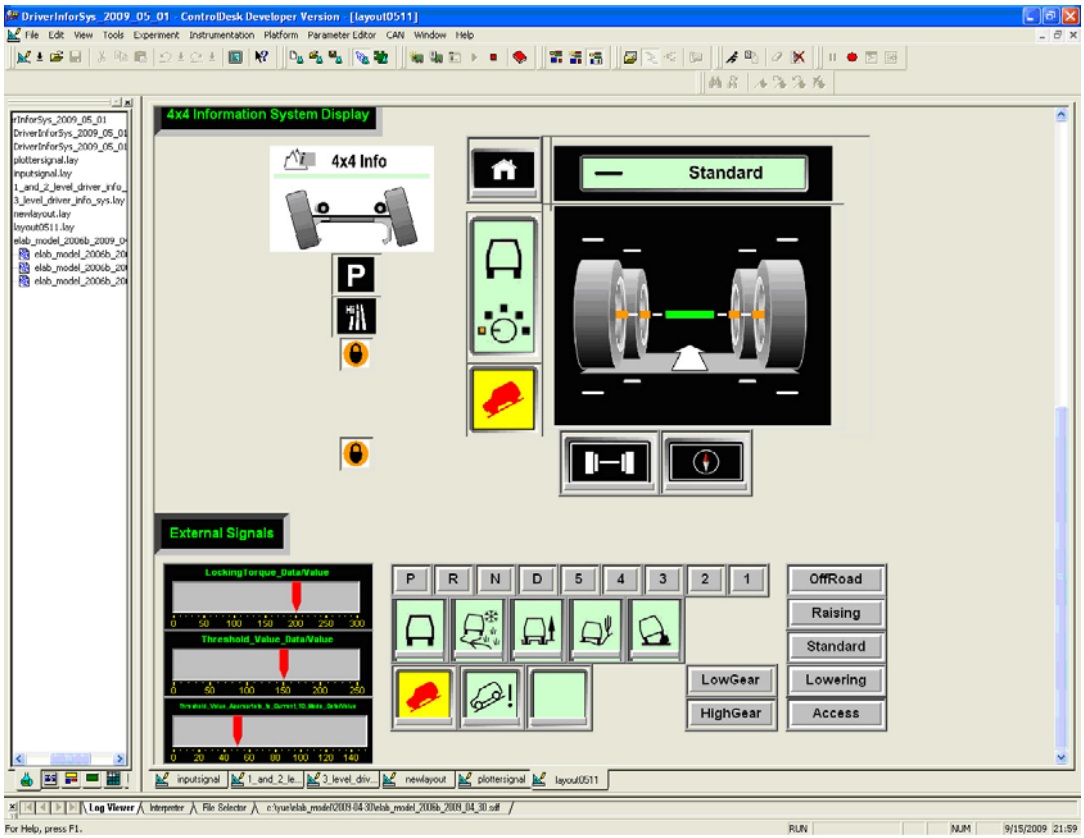


Fig. 6-14. The real-time animation of 4×4 Information System interface.

Fig. 6-14 illustrates the real-time animation of the 4×4 Information System. The top half of the window represents the 4×4 Information System display. It translates the design requirements into the visualized layout. Each block exhibits a type of information that can be viewed by the driver. The control blocks are integrated at the bottom of this layout. The sliders can be dragged to simulate different input values. The vehicle settings can be changed and driving modes can be switched by clicking buttons. This simulates the possible behaviour reactions of a driver when they face different driving situations. The buttons on the display can be clicked to simulate different situations without any interruption to the experiment. Most importantly, the change of pictures represents various modes or settings of the vehicle on the real-time basis. Thus, the robustness and the reliability of designed and developed functions are ensured.

6.5 Discussion

Given the complexity of the automotive SoS such as the 4×4 Information System presented in Chapter 3, the software function of this SoS has to be developed in consideration of the physical structure of the vehicle network, i.e., the large amount of real-time data is captured with a large number of interactions among the electronic systems on the vehicle and they have to be delivered and displayed correctly in order to enable the advanced functions of the 4×4 Information System. Thus, the real-time behaviour of the software is vital to the success of flawless software delivery for the automotive electronic SoS development. This chapter proposes a novel approach to verify the advanced function of automotive electronic SoS through real-time simulation and animation.

In this chapter, RTW produces the C code from the Simulink/Stateflow model for the real-time platform target to implement the real-time animation of the 4×4 Information System interface. Experience shows that dSPACE ControlDesk provides the features for the verification of advanced functions. Developing the function model in Simulink/Stateflow and then generating the code and transferring the model into dSPACE ControlDesk to perform real-time animation is a feasible and effective approach to the development of an automotive electronic SoS. This technique improves confidence in the function model built and generated code. Moreover, the real-time animation helps developers to become aware of the complexity of the SoS and allows engineers to realize the function interface and execute their concepts in the very early stages of the development. They are significant benefits that ensure the successful development of an automotive electronic SoS.

Chapter 7

Conclusion and future work

A near exponential growth of in-vehicle, embedded electronic systems has been witnessed during the past 30 years. It has been driven by the premium automobile sector where, presently, electronics and software account for around 40% of the value of some vehicles. Current in-vehicle electronic systems have evolved from single standalone computer systems to distributed systems including several networks, large numbers of sensors, actuators and up to 50, or more, ECUs which are distributed throughout a vehicle. In systems terms, automotive embedded electronic systems can now be classified as a SoS. The design, implementation and management of such complex distributed systems, and their integration into one cohesive and reliable SoS brings new challenges for the automotive industry.

Against this background, it is necessary to develop new methodologies for capturing the requirements for the SoS at the outset of the product development process and conveying the requirements through the stages in the product development process.

Firstly, this Thesis presents a brief discussion of SE and SoSE. SE encourages the use of tools and methods to better comprehend and manage the complexity in

systems. Models play important and diverse roles to address “three evils” in systems engineering, namely, complexity, a lack of understanding and communication issues. Building the model can allow engineers to identify complexity, aid understanding and improve communication. In addition, model-based design which is known as the V-model integrates modelling into a design, development and validation process that can be applied to a number of different tools and methodologies. It has been proved very successfully in performing the role of designing, developing, and deploying new equipment or systems to satisfy specific requirements. SoSE has to be carried out under some level of uncertainty as it involves factors in multiple levels and domains. In other words, SoSE seeks to optimize a network of various systems brought together to meet specific needs. Consequently, model-based design with new techniques such as new modelling languages and tools has been investigated as a potential methodology to address the challenges in automotive electronic SoS development.

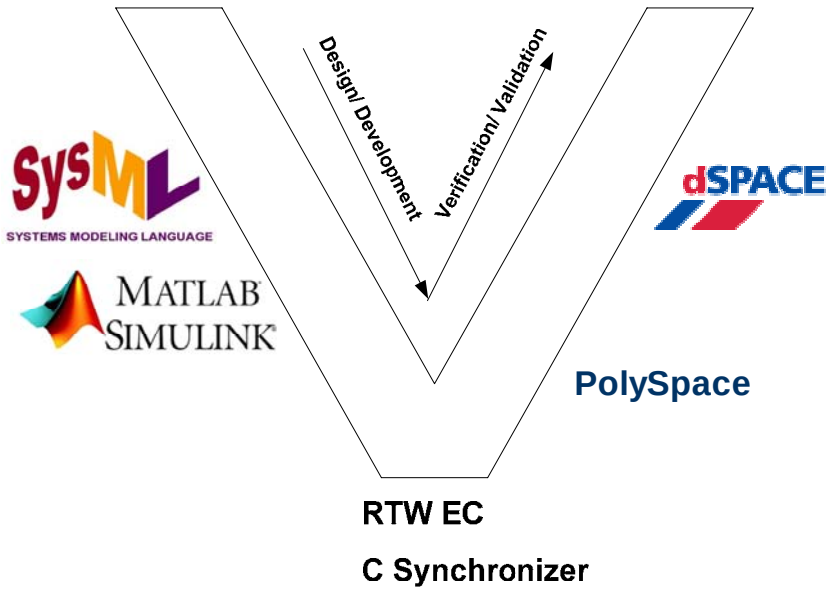


Fig. 7-1. The model-based design of the 4×4 Information System.

Fig. 7-1 shows the techniques which are investigated within the model-based design in this Thesis. The interaction and integration of the systems in the automotive electronic SoS has to evolve to accommodate the increasing complexity. The modelling languages and tools play an important role in the development of reliable and robust automotive electronic SoS in terms of requirements capture, modelling, auto coding, code checking and the verification of the software functions. This Thesis described two distinct model-based approaches to the development of automotive electronic SoS. The first approach has involved the use of the SysML modelling language that has emerged as a language based on UML but better suited to provide support for the engineering of systems and SoS. The SysML based tool ARTiSAN Studio is utilized for structural modelling, functional modelling and code generation. Specifically, this Thesis has explored: the use of block definition and internal block diagrams for structural modelling of the SoS; and use case, sequence, state machine and activity diagrams, for modelling the functional behaviour of the SoS. The listed diagrams in the model provide a clearly structured visualization of the 4×4 Information System. Several types of diagrams represent the design requirements in both structural and behavioural viewpoints of the SoS with relevant concerns. This model facilitates the formulation of the textual form specification documents and avoids interpretation leeway. Moreover, experience shows that these diagrams can be broken down and developed to different levels to represent the interactions and detail at various levels. Benefiting from the modelling tool ARTiSAN Studio, the model successfully translates the textual form specification documents to C code and it also has the flexibility in automatic code generation. The code can be produced from the entire model or generated from a certain component of the model, such as state machine diagrams

for software delivery. The model facilitates collaboration of people with different backgrounds such as systems engineers and software engineers to design, implement and manage such complex distributed systems and integrate these systems into one robust and reliable SoS.

The second approach involves the use of the MATLAB based tools Simulink and Stateflow for functional modelling and auto coding. Experience has shown that a Stateflow diagram has a very similar mechanism to the state machine diagram in SysML to model the functionality of a 4×4 Information System. Diagrams can be broken down and developed at different levels to show the interaction and details among these levels. Furthermore, unlike the state machine diagram in ARTiSAN Studio, the location of states in a Stateflow diagram is not restricted. Lower level states can be moved freely within the rectangle of higher level states. Therefore, the position of each display state in the Simulink/Stateflow model corresponds to the actual HMI. The coding implementation of the Simulink/Stateflow model has shown that the C or C++ code can be generated automatically for different targets by selecting corresponding system target files.

The application of ECUs on the vehicle is constantly increasing to address the challenges of increasing complexity while developing and maintaining electronic applications within the automotive SoS. The reliability of the SoS is directly affected by the quality of the embedded software in the ECUs which is developed from the software code. As a consequence, automatic code generation and static analysis are highly important benefits that make the development of an automotive electronic SoS efficient and effective. Therefore, further investigation of functional modelling has focused on the comparison of quality and efficiency of the code.

The static analysis tool PolySpace has been utilized to analyze C code and relate potential defects found therein back to the design model from which the code was generated. Specifically, the PolySpace tool has been applied to verify and analyze the C code generated from state machine diagrams of the SysML model built in ARTiSAN Studio and Stateflow diagrams in the Simulink/Stateflow model. The result shows that ARTiSAN Studio has a more comprehensive coverage of the code generation. It is capable of producing the code from a component of the model to the whole model. In consideration of the quality and efficiency, the Simulink/Stateflow model has demonstrated that it performs better in producing high quality and efficient C code. As a result, to develop an automotive electronic system, building the structure and function model in SysML by using ARTiSAN Studio is an essential and effective way to gain a comprehensive understanding of the entire system. In dealing with software delivery, developing the function model in Simulink/Stateflow and automatically generating the code from such a model is the preferred and efficient approach that has been demonstrated in this Thesis.

The real-time behaviour of the embedded software is vital to the success of the flawless software delivery for the automotive electronic SoS development. Within an automotive electronic SoS such as the 4×4 Information System, a large amount of real-time data is captured among the networked electronic systems on the vehicle and they have to be delivered and displayed correctly in order to enable the advanced functions. Hence, the software functions are required to be tested on a real-time basis. In dealing with this demand, this Thesis provides a useful complement to the offline simulation of the Simulink/Stateflow model through the real-time simulation and animation. Specifically, this Thesis has examined the ability to easily construct a real-time simulation and animation of the 4×4

Information System by using dSPACE ControlDesk from the automatically generated C code in Simulink/Stateflow model to verify advanced function of automotive electronic SoS on a real-time basis. The experiment has demonstrated that dSPACE ControlDesk provides the features for real-time simulation, testing and animation within the advanced function development. Developing the function model in Simulink/Stateflow and then generating the code and transferring the model into the dSPACE ControlDesk to enable real-time animation is a feasible and effective approach to the development of an automotive electronic SoS.

The outcome of this research is about the model-based design of a particular SoS within an automotive electronic SoS, i.e. a 4×4 Information System. The major contribution of this Thesis to the automotive industry is that the proposed techniques in model-based design will potentially fulfil the requirement of capturing, designing and developing needs for the development of an automotive electronic SoS.

The collaboration and integration of currently adopted techniques are gaining increased attention such as coupling UML and Matlab/Simulink models through co-simulation and integration based on a common underlying executable language [126]. The direction for future research could focus on the investigation of the capabilities of emerging tools and techniques in the industry. For instance, the ARTiSAN Studio released by ARTiSAN Software Tools Inc. [127] in 2008 has stated that “ARTiSAN Studio includes a comprehensive synchronization tool for Simulink that provides an integration enabling users to easily move between and synchronize information that is captured in both tools”, “The synchronizer also allows the interface to the software generated from Simulink to be customized in Studio and synchronized back into Simulink”. The outcome of research will be

beneficial in order to refine the model-based development of the automotive electronic SoS.

References

[1] Robert Bosch GmbH, *Automotive Electrics Automotive Electronics*. Suffolk: Professional Engineering Publishing Limited, 2004.

[2] R. C. Lind, H. W. Yen and D. Welk, “Evolution of the car radio: from vacuum tubes to satellite and beyond,” SAE paper: 2004-21-0001, *SAE World Congress*, Oct. 2004.

[3] R. K. Jurgen, *Automotive electronics handbook.* McGraw-Hill, New York, 1999.

[4] N. Navet, Y. Q. Song, F. Simonot-Lion and C. Wilwert, “Trend in automotive communication systems,” *Proc. IEEE Special Issue Ind. Commun. Syst.*, vol. 93, Issue: 6, pp.1024-1223, Jun. 2005.

[5] G. Leen and D. Heffernan, “Expanding automotive electronic systems,” *IEEE Comput.*, vol. 35, no. 1, pp. 88-93, Jan. 2002.

[6] Robert Bosch GmbH, *Gasoline-Engine Management.* Bury St. Edmunds: Robert Bosch GmbH, 2004.

[7] A. Shrinath and A. Emadi, “Electronic control units for automotive electrical power systems: Communication and networks,” *Proc. IMechE Part D: J. Automobile Engineering*, vol. 218, pp. 1217-1230, Jun. 2004.

[8] MOST, *Media Oriented Systems Transport Specification V2.5.* [Online]. Available: http://www.mostnet.de

[9] G. Leen, D. Heffernan and A. Dunne, "Digital networks in automotive vehicle," *J. Comput. Control Eng.*, pp. 257-266, Dec. 1999.

[10] E. Ortega, T. Heurung and R. Swanson, "System design from wires to warranty," *Automotive Electronics Magazine*, pp. 14-18, Feb. 2006.

[11] R. McMurran, F. McKinney, N. J. Tudor and W. Milam, "Dependable Systems of Systems," SAE paper: 2006-01-0597, *SAE World Congress*, Michigan, Apr. 2006.

[12] J. Boardman and B. Sauser, "System of Systems – the meaning of 'of'," *Proc. IEEE International Conference on System of Systems Engineering*, Los Angeles, CA, USA, pp. 118-123, Apr. 2006.

[13] J. Marco and E. Cacciatori, "The use of model based design techniques in the design of hybrid electric vehicles," *The 3rd International IET Conference on Automotive Electronics*, University of Warwick, UK, Jun. 2007.

[14] J.-L. Boulanger and Van Quang Dao, "Experiences from a model-based methodology for embedded electronic software in automobile," *Proc. 3rd International Conference on Information and Communication Technologies*, pp. 1-6, Apr. 2008.

[15] J.-L. Boulanger and Van Quang Dao, "Requirements engineering in a model-based methodology for embedded automotive software Research," *Proc. IEEE International Conference on Innovation and Vision for the Future*, pp. 263-268, Jul. 2008.

[16] B. Murphy, A. Wakefield and J. Friedman , “Best practices for verification, validation, and test in model-based design,” SAE paper: 2008-01-1469, *SAE World Congress*, 2004.

[17] K. Grimm, “Software technology within an automotive company - major challenges,” *Proc. IEEE 25th International Conference on Software Engineering*, Portland, Oregon, USA, 2003.

[18] J. Holt and S. Perry, *SysML for Systems Engineering.* IET Books, London, 2008.

[19] G. Grassl and G. Winkler, “Model-based development with automatic code generation – challenges and benefits in a DCT high-volume project,” SAE paper: 2008-01-0745, *SAE World Congress*, Apr. 2008.

[20] M. Conrad, H. Dörr, "Model-based development of in-vehicle software." *Proc. Conference on Design, Automation and Test in Europe*, pp. 89-90, Mar. 2006.

[21] Object Management Group, Unified Modelling Language. [Online]. Available: http://www.uml.org

[22] J. Holt, *UML for systems engineering: watching the wheels.* IEE publishing, London, 2004

[23] M. Hause, “The SysML modelling language,” *Proc. 5th European Systems Engineering Conference,* Sep. 2006

[24] H. Giese, S. Henkler, “A survey of approaches for the visual model-driven development of next generation software-intensive systems,” *J. Visual Languages and Computing*, vol. 17, pp. 528-550, 2006.

[25] Object Management Group, *Systems Modeling Language final adopted specification*. [Online]. Available: http://www.omgsysml.org

[26] Object Management Group. [Online]. Available: http://www.omg.org

[27] T. Weilkiens, *Systems engineering with SysML/UML: modelling, analysis, design.* Morgan Kaufmann Publishers and the Object Management Group, USA, 2007.

[28] MATLAB, Simulink, Stateflow, The MathWorks Inc. [Online]. Avaiable: http://www.mathworks.com/products

[29] Artisan Software. [Online]. Avaiable: http://www.artisansw.com

[30] PolySpace Technologies . [Online]. Avaiable: http://www.polyspace.com

[31] dSPACE GmbH. [Online]. Available: http://www.dspace.de

[32] J. A. Hoffer, J. F. George and J. Valacich, *Modern Systems Analysis and Design,* Pearson Education International, New Jersey, 2005.

[33] N. Fenton and G. Hill, *Systems Construction and Analysis,* McGraw-Hill Book Company, London, 1993.

[34] J. L. Shearer, A. T. Murphy and H. H. Richardson, *Introduction to System Dynamics,* Addison-Wesley Publishing Company, London, 1991.

[35] O. Katsuhiko, *System dynamics,* Pearson/Prentice Hall, NJ, 2004.

[36] S. J. Luskasik, "Systems, systems of systems, and the education of engineers," *J. Artificial Intelligence for Engineering Design, Analysis, and Manufacturing*, Vol. 12 (1), pp. 55-60, 1998.

[37] International Council of Systems Engineering (INCOSE), *Systems Engineering Handbook,* Version 3.1, August 2007. [Online]. Available: http://www.incose.org

[38] P. Sage, "Systems engineering education," *IEEE Trans. Systems, Man, and Cybernetics, Part C: Applications and Reviews*, vol. 30 (2), pp. 164-174, May. 2002.

[39] H. H. Goode and R. E. Machol, *System Engineering*, McGraw-Hill, New York, 1957.

[40] R. C. Booton and S. Ramo, **"**The development of systems engineering," *IEEE Trans. Aerospace and Electronic Systems*, vol. AES-20, no. 4, pp. 306-309, 1984.

[41] "What is systems engineering?" *IEEE Aerospace and Electronic Systems Magazine*, Vol. 15, Issue 10, pp. 9-10, Oct. 2000.

[42] S. Burnham, "Systems engineering: a practical approach for junior engineers," IEEE Aerospace and Electronic Systems Magazine, Vol. 21, Issue 6, Part 1, pp.3-8, Jun. 2006.

[43] A. W. Wymore, *Model-based Systems Engineering*, CRC Press, 1993.

[44] E. Aslaksen and R. Belcher, *Systems Engineering*, Prentice-Hall, 1992.

[45] Core Courses, "Systems analysis - architecture, behavior and optimization". Cornell University. [Online]. Available: http://systemseng.cornell.edu/CourseList.html

[46] M. Kayton, "A practitioner's view of system engineering," *IEEE Trans. Aerospace and Electronic Systems*, Vol. 33 (2) Part 2, pp. 579 – 586, Apr. 1997.

[47] A. Asbjornsen and R. J. Hamann, "Toward a unified systems engineering education," *IEEE Trans. Systems, Man, and Cybernetics, Part C: Applications and Reviews*, Vol.30 (2), pp. 175-182, May. 2000.

[48] Shenhar, "Systems engineering management: a framework for the development of a multidisciplinary discipline," *IEEE Trans. Systems, Man and Cybernetics*, Vol. 24 (2), pp. 327-332, Feb. 1994.

[49] NASA, “System analysis and modeling issues," *NASA Systems Engineering Handbook,* pp.85-98, Jun. 1995.

[50] J. A. Lane, “Process Evolution to support system of systems engineering,” *Proc. ULSSIS,* Germany, 2008.

[51] J. Ring and A. Madni, “Key challenges and opportunities in 'system of systems' Engineering,” *Proc. IEEE International Conference on Systems, Man and Cybernetics,* pp. 973–978, Oct. 2005.

[52] K. Cureton and F. Stan Settlers, “System-of-systems architecting: educational findings and implications,” *Proc. IEEE International Conference on Systems, Man and Cybernetics,* pp. 2726-2731, Oct. 2005.

[53] C. Keating, “Research foundations for system of systems engineering,” *Proc. IEEE International Conference on Systems, Man and Cybernetics,* pp. 2720–2725, Oct. 2005.

[54] A. Sousa-Poza, S. Kovacic and C. Keating, “System of systems engineering: an emerging multidiscipline,” *Int. J. System of Systems Engineering,* vol. 1, No. 1-2, pp. 1-17, 2008.

[55] P. Chen and J. Clothier, “Advancing systems engineering for system of systems challenges, ” *J. Systems Engineering,* vol. 6, No. 3, pp. 170-183, 2003.

[56] C. Keating, R. Rogers, R. Unal and D. Dryer, “System of systems engineering,” *J. Engineering Management,* Vol. 15 (3), pp. 36-45, Sep. 2003.

[57] C. Keating, A. Sousa-Poza, and J. Mun, “Toward a methodology for system of systems engineering,” *Proc. American Society of Engineering Management,* pp. 1-8, 2003.

[58] J. O. Clark, "System of systems engineering and family of systems engineering from a standards perspective," *Proc. IEEE International Conference on System of Systems Engineering*, pp. 1-6. Jun. 2008.

[59] J. O. Clark, "System of systems engineering and family of systems engineering from a standards, V-model, and dual-V model perspective," *Proc. 3rd Annual IEEE International Systems Conference*, pp. 381-387. Vancouver, Mar. 2009.

[60] IEEE 1220. 1994. *IEEE Trial-Use Standard for Application and Management of the Systems Engineering Process.* Copyright IEEE. [Online]. Available: http://shop.ieee.org/ieeestore

[61] IEEE 1220. 1998 and 2005. *IEEE Standard for Application and Management of the Systems Engineering Process.* Copyright IEEE. [Online]. Available: http://shop.ieee.org/ieeestore

[62] EIA/IS-632. 1994. *Systems Engineering.* Copyright © 1994, Government Electronics and Information Technology Association a Sector of the Electronic Industries Alliance. [Online]. Available: http://geia.org

[63] EIA-632. 1998. *Processes for Engineering a System.* Copyright © 1999, Government Electronics and Information Technology Association a Sector of the Electronic Industries Alliance. [Online]. Available: http://geia.org

[64] ISO/IEC 15288. 2002. *Systems engineering – System life cycle processes.* Copyright International Organization for Standardization (ISO), American National Standards Institute, 25 West 43rd Street, New York, NY 10036. (212) 642-4900. [Online]. Available: http://webstore.ansi.org.

[65] ISO/IEC 15288. 2008. *Systems and software engineering – System life cycle processes.* Copyright International Organization for Standardization (ISO),

American National Standards Institute, 25 West 43rd Street, ew York, NY 10036. (212) 642-4900. [Online]. Available: http://webstore.ansi.org.

[66] ISO/IEC TR 19760. 2003. *Systems engineering – A guide for the application of ISO/IEC 15288 (System life cycle processes).* Copyright International Organization for Standardization (ISO), American National Standards Institute, 25 West 43rd Street, New York, NY 10036. (212) 642-4900. [Online]. Available: http://webstore.ansi.org

[67] D. DeLaurentis, D. Fry, O. Sindiy and S. Ayyalasomayajula, "Modeling framework and lexicon for system-of-systems problems," *IEEE Trans. Systems, Man, and Cybernetics-Part A: Systems and Humans,* 2006.

[68] D. DeLaurentis, "Understanding transportation as a system of systems design Pproblem," *Proc. 43rd AIAA Aerospace Sciences Meeting*, AIAA-2005-0123, Reno, Nevada, Jan. 2005.

[69] J. Bortolazzi, "Challenges in automotive software engineering," *International ICSE workshop on Software Engineering for Automotive Systems*, keynote presentation, 2004.

[70] S. Gumbrich, "Embedded systems overhaul: it's time to tune up for the future of the automotive industry." *IBM Business Consulting Services*, Dec. 2004.

[71] E. Ortega, T. Heurung and R. Swanson, "System design from wires to warranty," *Automotive Electronics Magazine*, pp. 14-18, Feb. 2006

[72] A. W. Wymore, *Model-based Systems Engineering,* CRC Press, 1993.

[73] M. Mutz, M. Huhn, U. Goltz and C. Kroemke, "Model based system development in automotive", SAE paper: 2003-01-1017 *SAE World Congress*, Mar. 2003.

[74] J. Schauffele and T. Zurawka, *Automotive Software Engineering: Principles, Processes, Methods, and Tools*, SAE International, USA, 2005.

[75] G. Hodge, J. Ye and W. Stuart, "Multi-target modeling for embedded software development for automotive applications," SAE paper: 2004-01-0269, *SAE World Congress*, 2004.

[76] G. Sandmann and R. Thompson, "Development of AUTOSAR software components within model-based design," SAE paper: 2008-01-0383, *SAE World Congress*, 2008.

[77] M. Weber, "Requirements engineering in automotive development - experiences and challenges," *Proc. IEEE Joint International Conference on Requirements Engineering*, Essen, Germany, 2002.

[78] S. Robertson and J. Robertson, *Mastering the Requirements Process*, Addison Wesley, London, 1999.

[79] K. Wiegers, *Software Requirements*, Microsoft Press, Redmond, 1999.

[80] I. Sommerville and P. Sawyer, *Requirements Engineering: A Good Practice Guide*, Wiley, Chichester, 1997.

[81] G. Kotonvy and I. Sommerville, *Requirements Engineering*, Wiley, Chichester, 1998.

[82] B. L. Kovitz, *Practical Software Requirements*, Manning, Grenwich, 1999.

[83] D. Gause and G. Weinberg, *Exploring Requirements: Quality Before Design*, *Dorset House*, New York, 1989.

[84] A. J. Kornecki, K. Hall, D. Hearn, H. Lau and J. Zalewski, "Evaluation of software development tools for high assurance safety critical systems," *Proc. 8th IEEE International Symposium on High Assurance Systems Engineering*, pp. 273 – 274, 2004.

[85] B. Berard, M. Bidoit, A. Finkel, F. Laroussinie, A. Petit, Petrucci, L. P. Schnoebelen, *Systems and Software Verification: Model-Checking Techniques and Tools,* Springer, New York, 2001.

[86] L. Vitkin and T. K. Jestin, "Incorporating autocode technology into software development process", *Proc. ICSE,* pp.51-57, 2004.

[87] P. Schubert, L. Vitkin, F. Winters, "Executable Specs: what makes one, and how are they used?" SAE paper: 2006-01-1357, *SAE World Congress*, 2006.

[88] W. Everett, S. Keene and A. Nikora, "Applying software reliability engineering in the 1990s," *IEEE Transactions on Reliability*, vol. 47 (3), pp. 372–378, Sep. 1998.

[89] S. Xiao and C. H. Pham, "Performing high efficiency source code static analysis with intelligent extensions," *Proc. APSEC,* pp. 346–355, 2004.

[90] Q. Systems, "Overview large java project code quality analysis," *QA Systems, Tech. Rep.*, 2002.

[91] J. Viega, J. T. Bloch, Y. Kohno and G. McGraw, "Its4: A static vulnerability scanner for c and c++ code," *Proc. IEEE 16th Annual Computer Security Applications Conference*, pp. 257-266, Washington, DC, USA, 2000.

[92] V. B. Livshits and M. S. Lam, "Finding security vulnerabilities in java applications with static analysis," *Proc. 14th USENIX Security Symposium*, 2005.

[93] A. Rai, "On the role of static analysis in operating system checking and runtime verification," technical Report FSL-05-01, *Stony Brook University, Tech. Rep.,* May. 2005.

[94] C. Artho, "Finding faults in multi-threaded programs," *Master's thesis,* Federal Institute of Technology, 2001. [Online]. Available:

http://citeseer.ist.psu.edu/artho01finding.html

[95] R. L. Glass, "The realities of software technology payoffs," *Commun. ACM*, vol. 42 (2), pp. 74–79, 1999.

[96] R. L. Glass, "Inspections - some surprise findings," *Commun. ACM*, vol. 42 (4), pp. 17–19, 1999.

[97] PolySpace Technologies, *PolySpace for C Documentation*, 2004.

[98] K. Kratkiewicz and R. Lippmann, "Using a diagnostic corpus of C programs to evaluate buffer overflow detection by static analysis tools," *2005 Workshop on the Evaluation of Software Defect Detection Tools*, Chicago, IL 2005.

[99] M. Zitser, Securing Software: An Evaluation of Static Source Code Analyzers, *Master's Thesis*, Massachusetts Institute of Technology, Cambridge, MA, 2003.

[100] M. Zitser, R. Lippmann and T. Leek, "Testing static analysis tools using exploitable buffer overflows from open-source code," *Proc. 12th ACM SIGSOFT International Symposium on Foundations of Software Engineering*, Newport Beach, CA, pp. 97-106, 2004.

[101] P. Emanuelsson and U. Nilsson, "A comparative study of industrial static analysis tools (Eextended version)," *Technical reports in Computer and Information Science* Report number 2008:3, Jan. 2008.

[102] J. Rumbaugh, I. Jacobson and G. Booch, *The Unified Modeling Language reference manual*. 2nd edn. Boston, MA, Addison-Wesley, 2005.

[103] M. Hause, F. Thom and A. Moore, "Inside SysML," *J. Computing & Control Engineering* Vol.16 (4), pp. 10-15, Sep. 2005.

[104] Land Rover, *"Driver Information System Handbook"*, Publication Part No.LRL 10 95 55 502, 2004.

[105] R. Rajamani, *Vehicle Dynamics and Control*, Springer, New York, United States, 2006.

[106] W. Lawrenz, *CAN System Engineering: From Theory to Practical Applications.* New York: Springer-Verlag, 1997.

[107] R. Land, "Applying the IEEE 1471-2000 Recommended Practice to a Software Integration Project," *Proc. International Conference on Software Engineering Research and Practice,* CSREA Press, 2003.

[108] L. Bass, P. Clements and R. Kazman, *Software Architecture in Practice,* Addison-Wesley, 1998.

[109] J. Bosch, *Design & Use of Software Architectures,* Addison- Wesley, 2000.

[110] P. Clements, F. Bachmann, L. Bass, D. Garlan, J. Ivers, R. Little, R. Nord and J. Stafford, *Documenting Software Architectures: Views and Beyond*, Addison- Wesley, 2002.

[111] P. Clements, R. Kazman and M. Klein, *Evaluating Software Architectures: Methods and Case Studies,* Addison-Wesley, 2002.

[112] C. Hofmeister, R. Nord and D. Soni, *Applied Software Architecture*, Addison-Wesley, 2000.

[113] M.W.A. Steen, D.H. Akehurst, H.W.L. ter Doest, M.M. Lankhorst, "Supporting viewpoint-oriented enterprise architecture," *Proc. IEEE 8th International Conference on Enterprise Distributed Object Computing,* pp. 201-211, 2004.

[114] IEEE Architecture Working Group, *IEEE Std 1471-2000*, Recommended Practice for Architectural Description of Software-Intensive Systems. IEEE, USA, 2000.

[115] A. Cockburn, *Writing Effective Use Cases,* Addison Wesley, 2001.

[116] Y. Guo, A. Chakrapani Rao and R. P. Jones, "Architectural and functional modelling of an automotive Driver Information System using SysML," *Proc. 2008 IEEE/ASME International Conference on Mechatronic and Embedded Systems and Applications,* pp. 552-557, Beijing, China, Oct. 2008.

[117] Y. Guo and R. P. Jones, "A study of approaches for model based development of an automotive Driver Information System." *Proc. 2009 IEEE International Systems Conference,* pp. 267-272, Vancouver, British Columbia, Canada, Mar. 2009.

[118] A. F. Ackerman, L. S. Buchwalk and F. H. Lewski, "Software inspections: an effective verification process," *IEEE Software,* vol. 6 (3), pp. 31-36, May. 1989.

[119] V. D'Silva, D. Kroening, G. Weissenbacher, "A survey of automated techniques for formal software verification," *IEEE Trans. Computer-Aided Design of Integrated Circuits and Systems,* Vol. 27 (7), pp.1165 – 1178, Jul. 2008.

[120] T. Erkkinen, M. Conrad, "Safety-critical software development using automatic production code generation," *The MathWorks, Inc.* 2007.

[121] T. Erkkinen, C. Hote, "Automatic flight code generation with integrated static run-time error checking and code analysis," *AIAA Modeling and Simulation Technologies Conference and Exhibit,* Keystone Colorado, 2006.

[122] I. Stürmer, D. Weinberg, M. Conrad, "Overview of existing safeguarding techniques for automatically generated code," *Proc. 2nd Intl. ICSE Workshop on Software Engineering for Automotive Systems*, pp. 1-6, St. Louis, Missouri, USA, May. 2005.

[123] J. Zheng, L. Williams, N. Nagappan, W. Snipes, J. P. Hudepohl and M. A. Vouk, "On the value of static analysis for fault detection in software," *IEEE Trans. Software Engineering*, Vol. 32 (4), pp. 240 – 253, Apr. 2006.

[124] H. Hu, G. Xu, Y. Zhu, "Hardware-in-the-loop simulation of electric vehicle powertrain system," *Proc. Power and Energy Engineering Conference, APPEEC 2009*, pp.1-5, Mar. 2009.

[125] dSPACE GmbH, *dSPACE Catalog 2009*, pp.508-511, 2009.

[126] Y. Vanderperren, W. Dehaene, "From UML/SysML to Matlab/Simulink: current state and future perspectives," *Proc. Conference on Design, Automation and Test in Europe*, pp. 1-1, Mar. 2006.

[127] Artisan Software, *Simulink Integration.* [Online]. Available: http://www.artisansoftwaretools.com/products/tool-set/simulink-integration

Appendix A

Functionality of the 4×4 Information System

The functionalities of 4×4 Information System which is a part of the Driver Information System as shown in Fig. 3-3 of Chapter 3 are described in detail in this section.

Screen layout

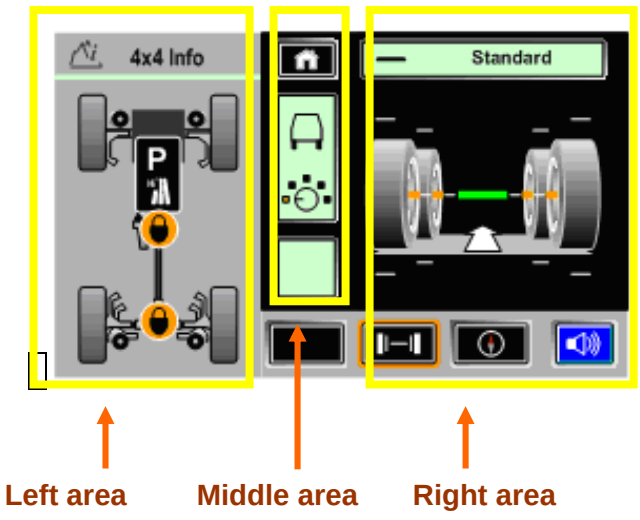


Fig. A-1. The 4×4 Information System display.

Fig. A-1 is the 4×4 Information System display of the vehicle. As shown in Fig. A-1, there are three areas in the 4×4 Information System display, i.e., left, middle and right. The left area is a permanent display area which displays the information on the chassis and powertrain of the vehicle, including steering angle information, high/low ratio selection status, gear position and differential lock information for both the centre and rear. The middle area is also a permanent display area showing the TO mode and HDC information. The right display area changes when different views are selected including the compass view and chassis view. To access the compass view and chassis view, the driver needs to correspondingly press the "Compass view" or "Chassis view" soft key on the touch screen which is highlighted in a red circle. The air suspension status and wheel height status are also displayed in this area within different views.

Primary functions

The primary functions are related to the vehicle status. On the screen, graphics created for the 4×4 Information System will be displayed and changed to reflect data which describe the state of the 4×4 Information System. As mentioned in Section 3.4.1 of Chapter 3, the chassis and powertrain information is shown in the left hand area. The functions on the left display area are

- Steering angle status
- Gearbox status
- Gear position
- Differential lock information

Fig. A-2 shows how steering angle data are displayed on the screen.

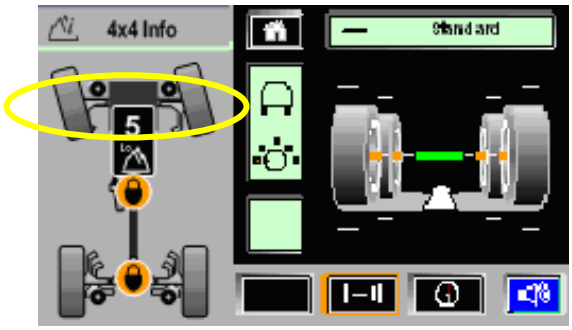


Fig. A-2. Steering angle data changes.

The maximum orientation of the wheel graphics is 30 degrees from the straight ahead position indicating full lock. The steered wheel images represent the steering angle data by displaying one of 13 graphical images of the steered wheels. Each image displays for a specific range of steering wheel angle data which is 5 degrees from -30 to +30 degrees. When the steering wheel data exceed the appropriate range for the current graphic, the appropriate steered wheel graphic should be updated for both of the steered wheel images concurrently.

The high and low range of transfer gear can be selected and represented graphically in the chassis map.

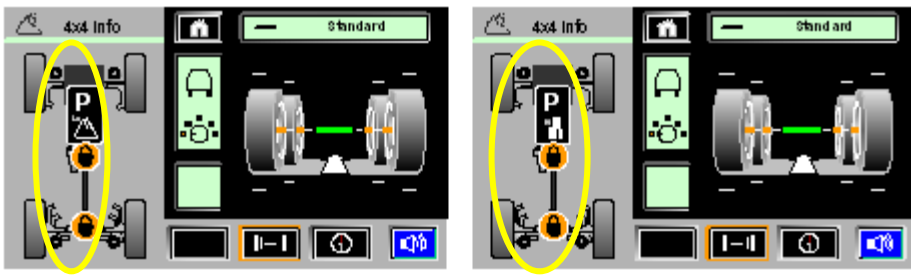


Fig. A-3. Transfer gearbox data.

When a range is selected, the appropriate graphic will be displayed in the chassis map. As shown on the left of Fig. A-3, when the transfer gearbox is successful in the low range, the "low range" icon is displayed in the chassis plan view. The display also shows the currently selected gear. When the parking gear is selected as shown in the Fig. A-3, the symbol “P” is displayed on the gearbox

graphic. For the other gear selections, the display will show “P R N D 5 4 3 2 1” respectively. The differential lock information for both the centre and rear is displayed under the gear selections by two icons showing “Locked” or “Unlocked”.

The air suspension status is displayed at the top right of screen. The air suspension has three suspension heights:

- Off-road
- Standard
- Access

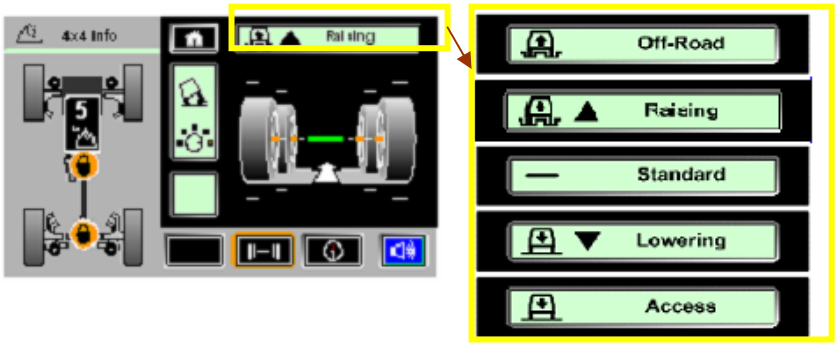


Fig. A-4. Air suspension status.

Fig. A-5. Control panel and buttons.

When the vehicle is in any of these states, the suspension status window in the top right of the display indicates the current suspension setting. As shown in Fig. A-4, the air suspension is rising to “Off-Road”. The air suspension status can be controlled manually by pushing the selector upwards or downwards which is next

to the HDC button as shown in Fig. A-5. The air suspension status can also be changed automatically according to the TO settings. In Fig. A-4, as soon as this height change is required, the display shows the text message “Raising” and replaces the previous height graphic. In addition, an arrow is displayed indicating the direction of vehicle travel. During a height change, the arrow head will flash. The wheel height graphical display will progressively change, showing the changing relationship between the individual wheels and the vehicle body. When the “Off-Road” height is reached the arrow icon disappears and the current vehicle height is displayed as “Off-Road”.

The air suspension setting is also displayed graphically in the wheel displacement window in the chassis view. To access the chassis view, the driver needs to press the “Chassis” soft key in the bottom right display area highlighted in the red rectangle. From the chassis view, the suspension status and wheel displacement status can be viewed.

The chassis view display window contains a representation of the four road wheels, along with several discrete graphical elements. These graphical elements move in direct response to actual wheel height changes. The vertical position of each road wheel graphic is determined by data from height sensors. As shown in Fig. A-4 the white dotted line shows the nominal vehicle height. When the vehicle is at either the off-road ride height or the access ride height, the white dotted line, which is the wheel height display, must configure to maintain the nominal height of the wheels relative to the ground plane. A solid green line between the wheels represents the practical height of the vehicle. When a wheel is at its standard height, the solid green line will line up with the white dotted line in the vertical axis. This is represented on the left of Fig. A-4. When vehicle is set to off-road ride height,

the air springs are extended to push the wheels further away from the chassis, which lifts the vehicle body by a controlled distance. If a vehicle has a specific off-road ride height that is 50mm above the standard ride height, then this would increase the data value of each wheel by 50mm. The green line moves up the screen in appropriate distance and in this case, it is 10 pixels for 50 mm of body movement.

Framing each of the wheel graphics are eight white lines that act as markers to indicate the extremes of each wheel travel. When a road wheel is at the extreme of its travel, the edge of the wheel graphic will line up with the appropriate travel limit marker. Four orange nodes on the road wheels indicate the position of the wheels associated with the vehicle height. Although the front and rear wheels appear to be different heights, they are only presented in this way to give a sense of perspective view. The graphical response of each wheel to change in wheel height data is exactly the same. When a wheel is at its standard height, the centre of the wheel will line up with the datum line in the vertical axis. Vehicles with air suspension will maintain a set ride height under all loading conditions up to the design loading limit. The suspension system will compensate for the increased load by increasing the air pressure in the system.

The above functions are related to vehicle powertrain and chassis. Besides, the vehicle can also provide five different TO settings and HDC to the driver. Their status will be displayed on the HLDF by the 4×4 Information System. TO settings and HDC are displayed in the middle area of the screen.

Fig. A-6. TO settings button.

As shown in Fig. A-6, driver can select 5 different TO modes by rotating the button on the vehicle:

- Standard
- Grass / Snow / Ice
- Mud / Ruts
- Sand
- Rock crawl

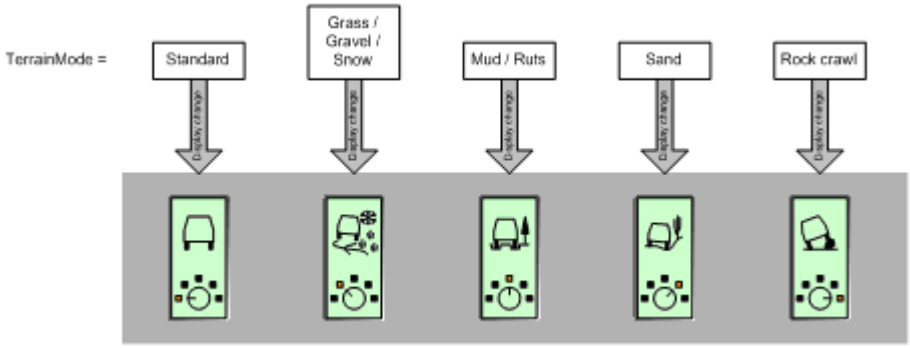


Fig. A-7. TO settings display.

As shown in Fig. A-7, the appropriate vehicle icon for the currently active TO mode should be shown in the central TO display window. When the driver shifts from different TO mode, the engine, transmission, suspension and traction settings are all reconfigured to deliver the best possible off-road driving to the driver.

HDC is used to provide a smooth and controlled hill descent in rough terrain without the driver needing to touch the brake pedal. After the driver pushes the

HDC button, which is in yellow as shown in Fig. A-6, the vehicle will descend using the ABS to control the speed for each wheel. If the vehicle accelerates without pushing the accelerator pedal, the system will automatically apply the brakes to slow down the vehicle. Applying pressure to the accelerator or brake pedal will override the HDC system as the driver requires.

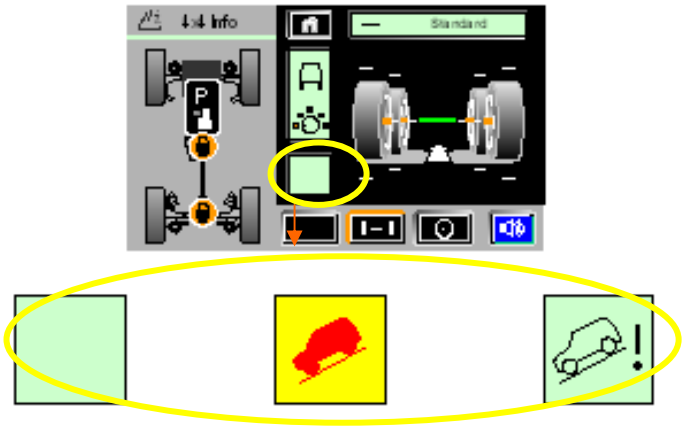


Fig. A-8. HDC.

The HDC status is highlighted in yellow as shown in Fig. A-8. The HDC system reports three states relating to its function:

- Inactive
- Set
- Pending

When the HDC is inactive, there is no display for the HDC function icon on the 4×4 display which is shown at the bottom as in Fig. A-8. When the HDC system is selected and activated, the red and yellow HDC function icon will be displayed continuously as shown at the middle bottom. When the HDC is selected, but there is a condition that inhibits the activation of the HDC such as wrong gear selected, the HDC icon on the display will flash.

Secondary functions

Besides the above primary functions, the 4×4 Information System can also provide a compass view for the driver. To access the compass view, the “Compass” soft key needs to be selected on the touch screen which is highlighted in an orange rectangle. The compass view window replaces the display of the wheel height information that is only displayed in the chassis view.

Fig. A-9. Compass view.

From Fig. A-9, the compass screen displays a graphic indicating the heading of the vehicle against the compass points. If the ‘North-up’ display mode is active in the navigation system, the compass points are fixed and the vehicle pointer will rotate to indicate the vehicle heading. If the “Heading up” display mode is active in the navigation system, then the vehicle pointer will be fixed vertically on the display and the compass points will rotate to indicate the vehicle heading.

An example of an off-road driving scenario

Fig. A-10 shows a scenario of off-road driving. The actual view can also be presented to the driver in the vehicle.

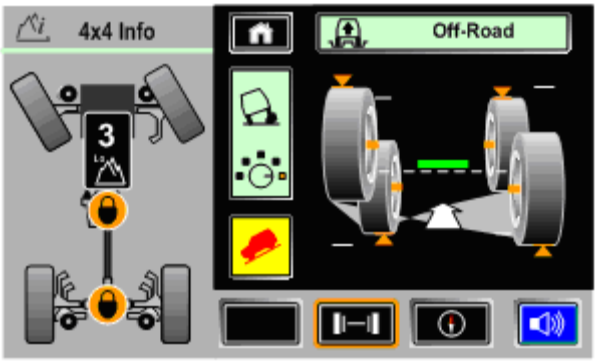


Fig. A-10. A scenario of off-road driving.

From this figure, the information provided to the driver can be viewed. The main gearbox is in 3rd gear, the transfer gearbox is in the low range, the steering wheel is at full left lock, the differential is locked, the TO setting is in rock crawl mode, the HDC is active and vehicle is in the off-road ride height. As we can see from the chassis view, the left rear wheel travels over a rock or similar obstacle. It is pushed up into the vehicle body. The data from the wheel height sensor are represented by moving the vertical position of the left rear wheel graphic up in the screen.

Appendix B

Diagrams in the model built in ArtiSAN Studio

This appendix provides a full list of diagrams in the model which is built in ARTiSAN Studio.

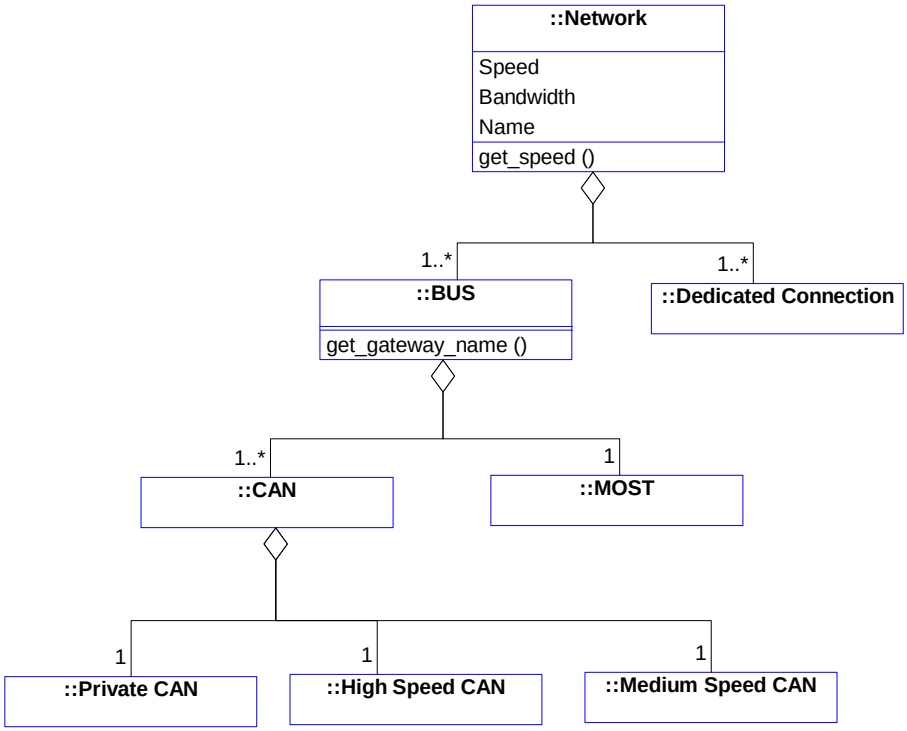


Fig. B-1. Block definition diagram 1: network class.

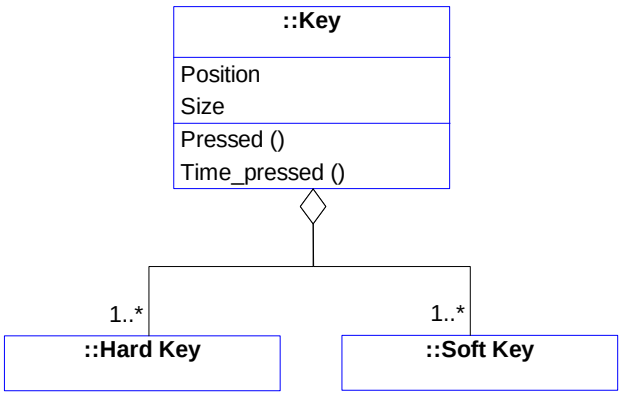


Fig. B-2. Block definition diagram 2: key class.

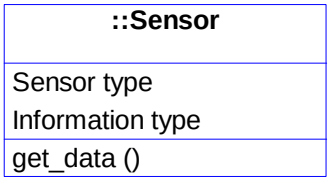


Fig. B-3. Block definition diagram 3: sensor class.

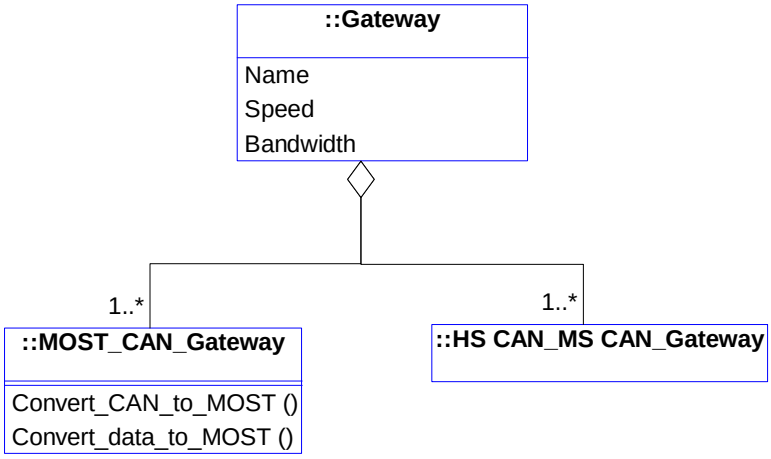


Fig. B-4. Block definition diagram 4: gateway_class.

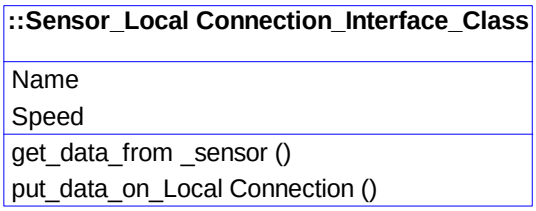


Fig. B-5. Block definition diagram 5: sensor_local connection_interface_class.

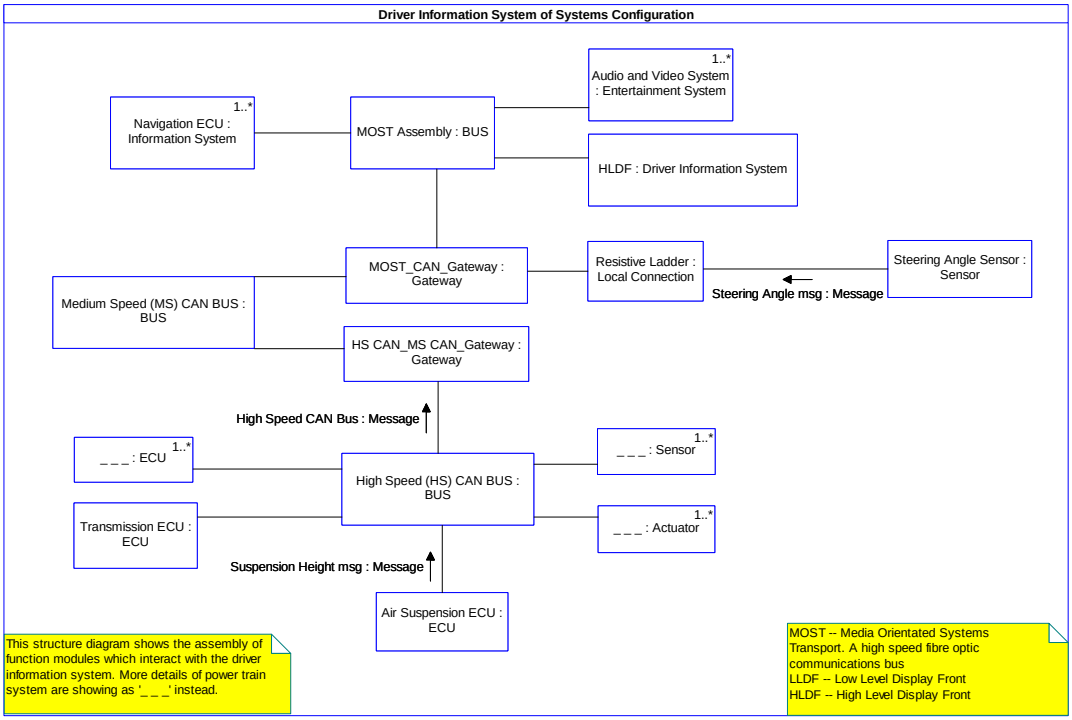


Fig. B-6. Internal block diagram 1: Driver Information System of Systems overview.

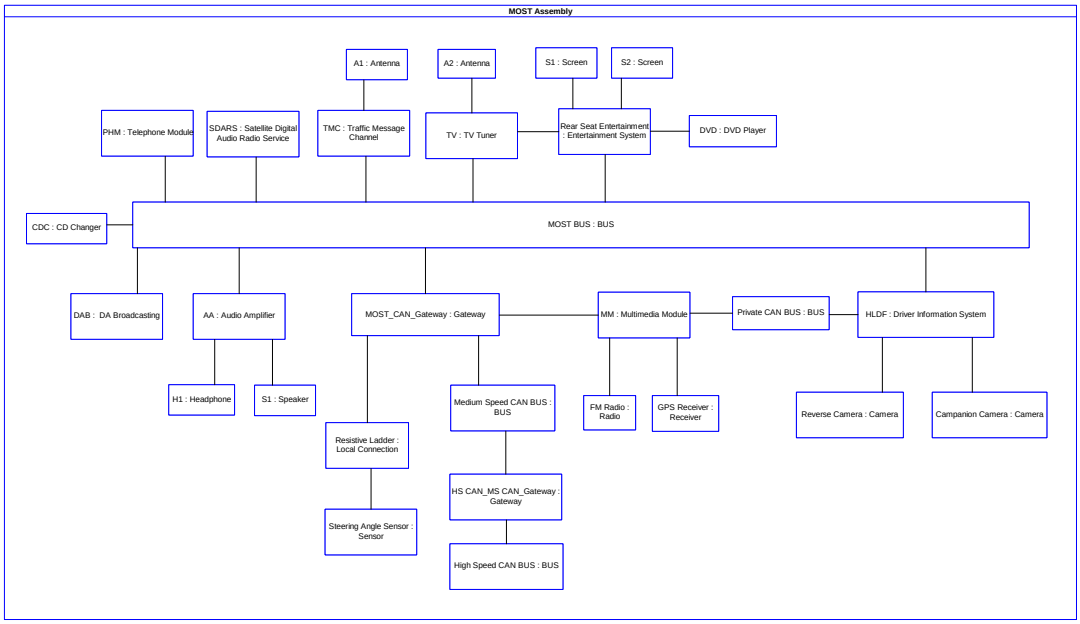


Fig. B-7. Internal block diagram 2: MOST System of Systems overview.

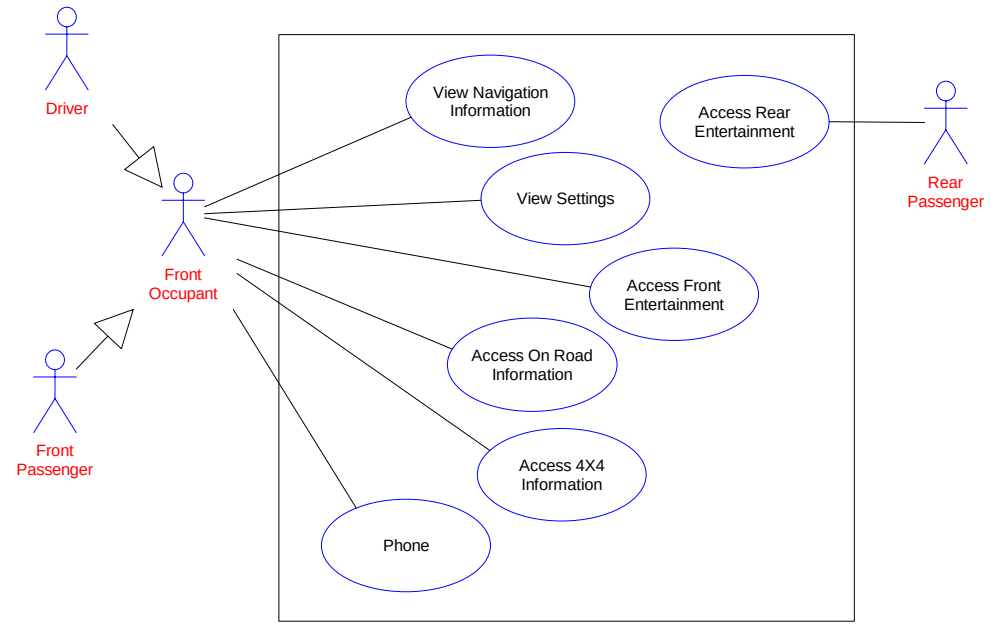


Fig. B-8. Use case diagram 1: Driver Information System use case.

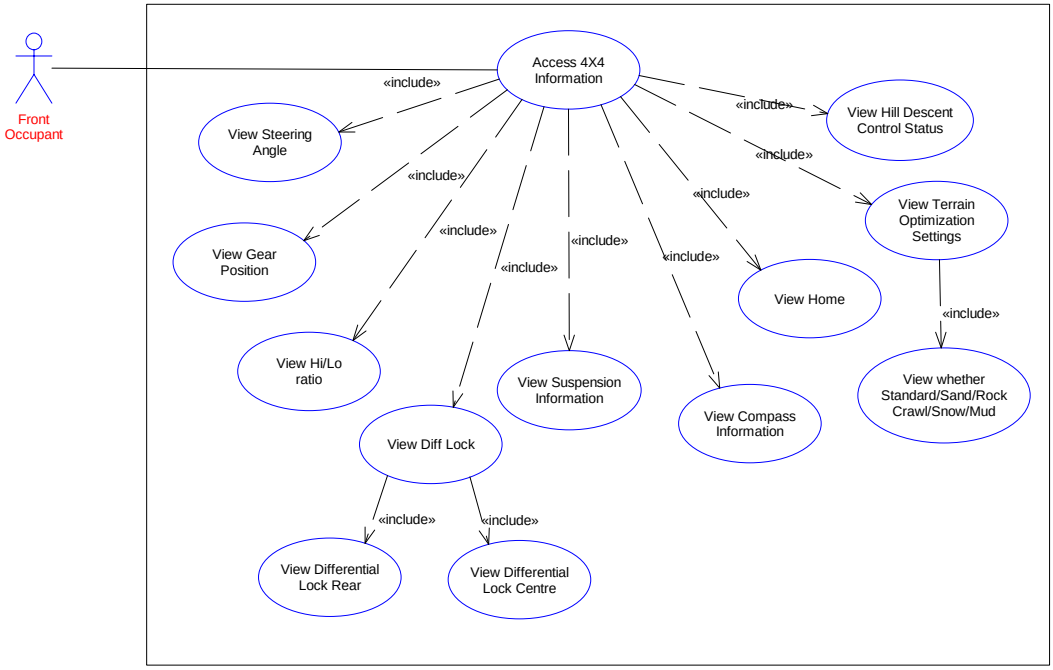


Fig. B-9. Use case diagram 2: 4×4 information use case.

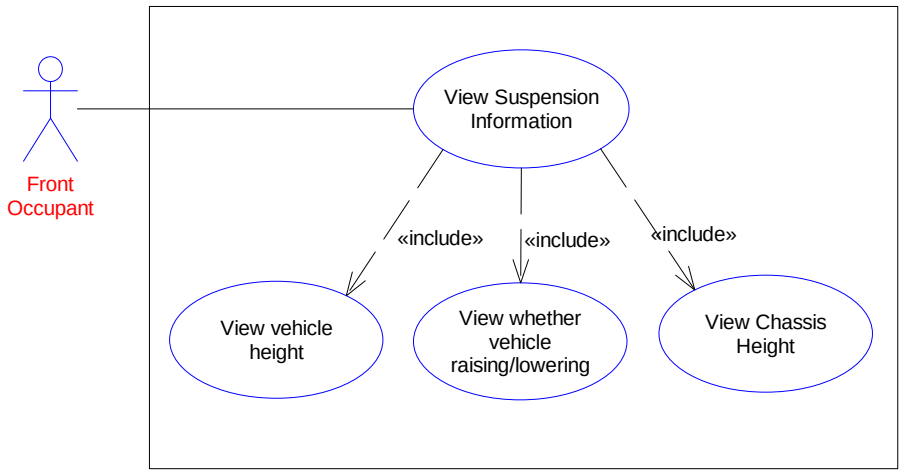


Fig. B-10. Use case diagram 3: suspension information use case.

Text Diagram 1.

Description of the Display layout

There are three areas in the driver information system display, left, central and right. The left area is a permanent display area which display chassis and power train of the vehicle with the following information list below:

Steering Angle Information,
High/Low ratio selection status,
Differential lock Information for both centre and rear,
Gear Position.

The central area contains the Home soft key and a permanent display area which shows the following information:

Terrain Optimization Mode,
HDC information.

The right display area changes when different views are selected by press soft keys at the bottom including Compass view and Chassis view.

Fig. B-11. Text diagram 1: layout of the display.

Text Diagram 2.

Air suspension selector

Air suspension selector is next to the Hill Descent Control button. The air suspension status can be controlled manually by pushing upwards or downwards the air suspension selector.

Fig. B-12. Text diagram 2: air suspension selector.

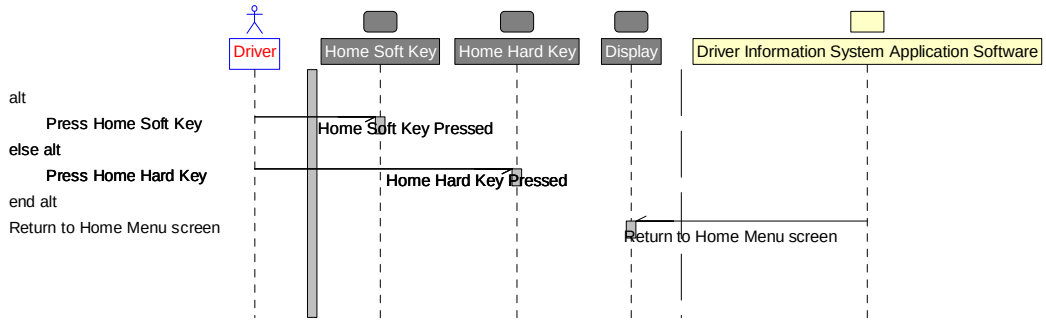


Fig. B-13. Sequence diagram 1: return to home menu screen from other screens (HL - B).

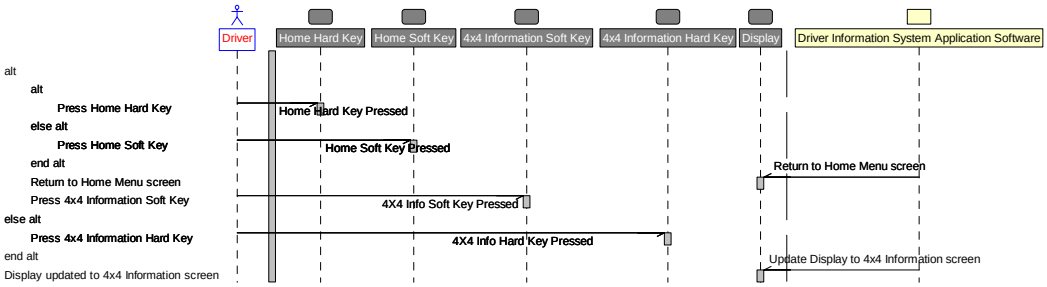


Fig. B-14. Sequence diagram 2: change to 4×4 information screen from other screens (HL - B).

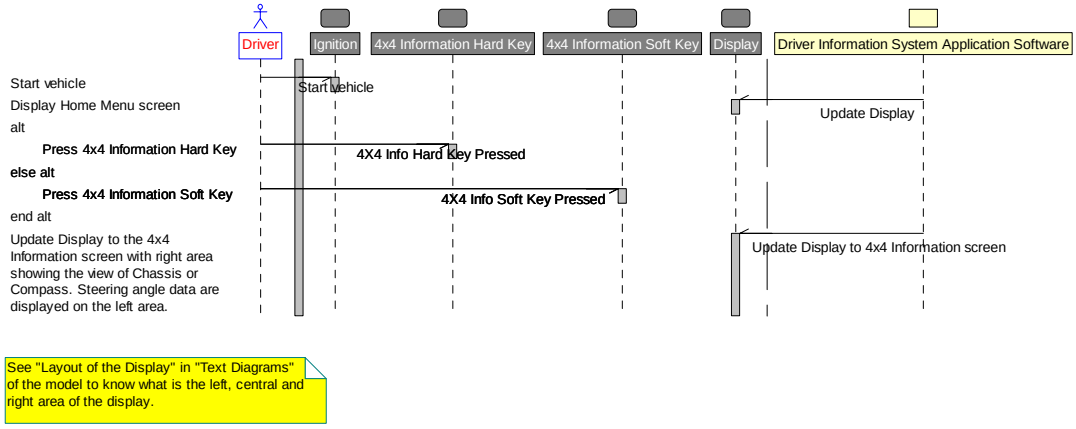


Fig. B-15. Sequence diagram 3: view steering angle (HL - B).

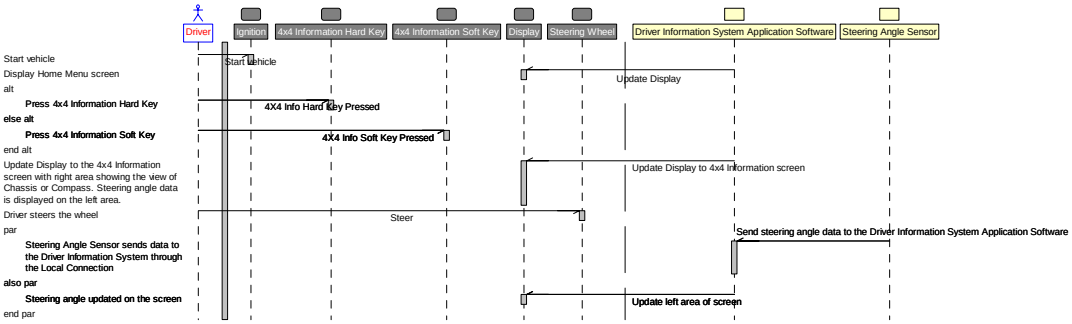


Fig. B-16. Sequence diagram 4: view steering angle (HL - W).

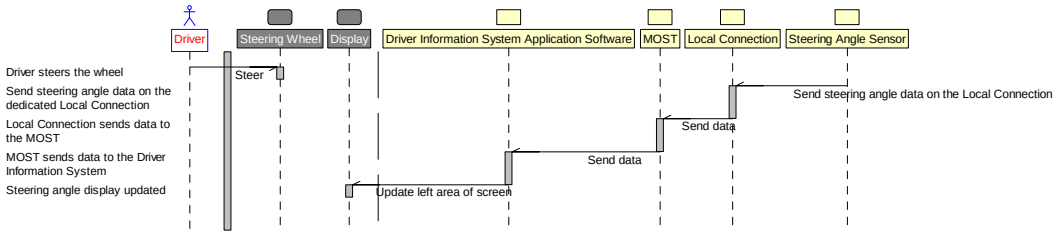


Fig. B-17. Sequence diagram 5: view steering angle (DL - B).

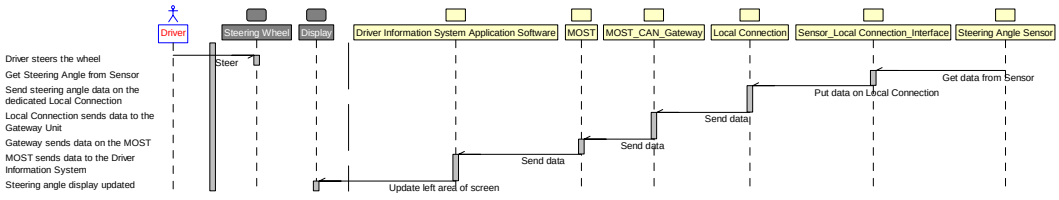


Fig. B-18. Sequence diagram 6: view steering angle (DL - W).

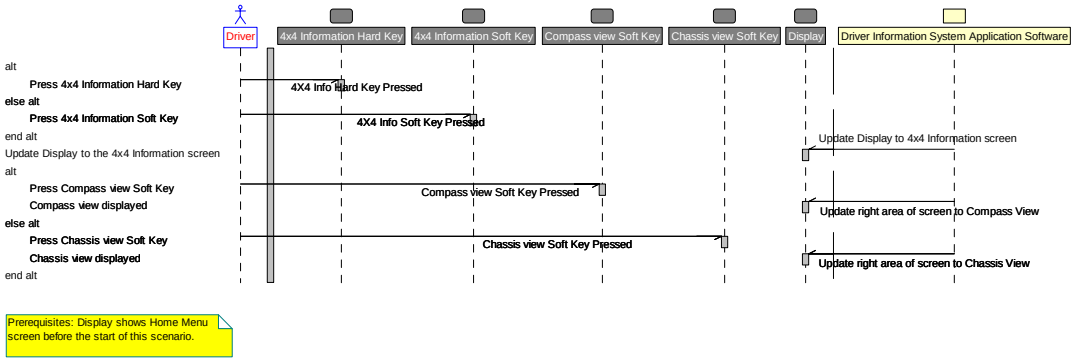


Fig. B-19. Sequence diagram 7: choose different views in 4×4 information screen from home menu screen (HL - B).

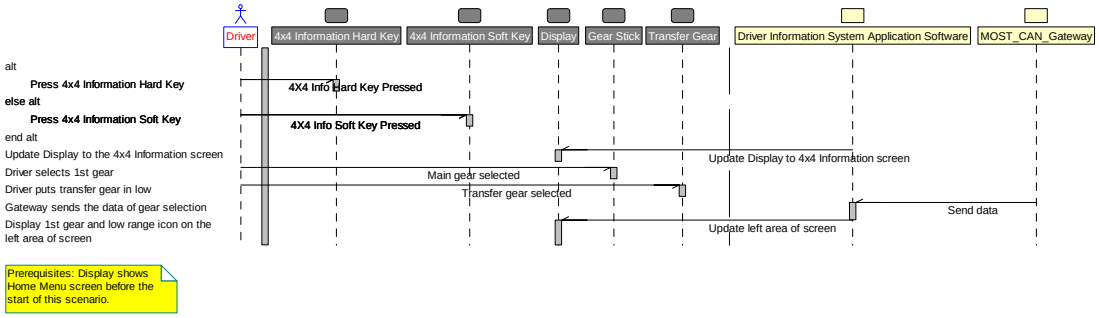


Fig. B-20. Sequence diagram 8: view main gear and transfer gear change from home menu screen (HL - B).

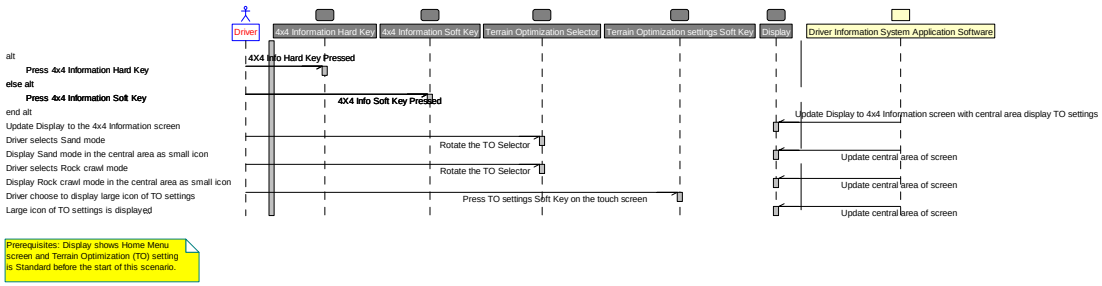


Fig. B-21. Sequence diagram 9: view terrain optimization (TO) settings from home menu screen (HL - B).

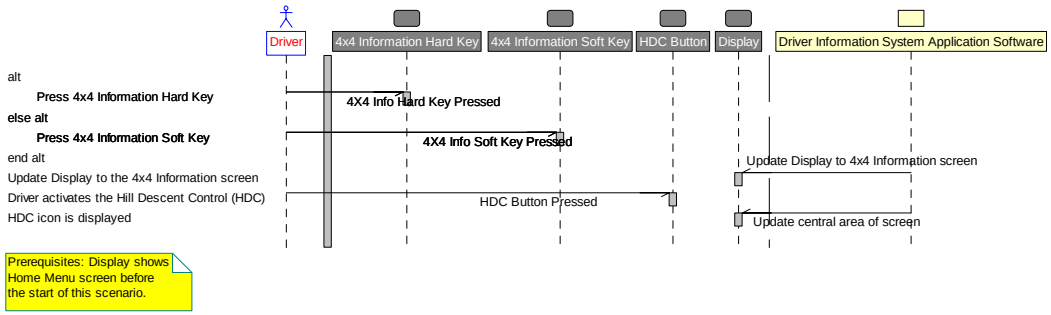


Fig. B-22. Sequence diagram 10: display of hill descent control (HDC) from home menu screen (HL - B).

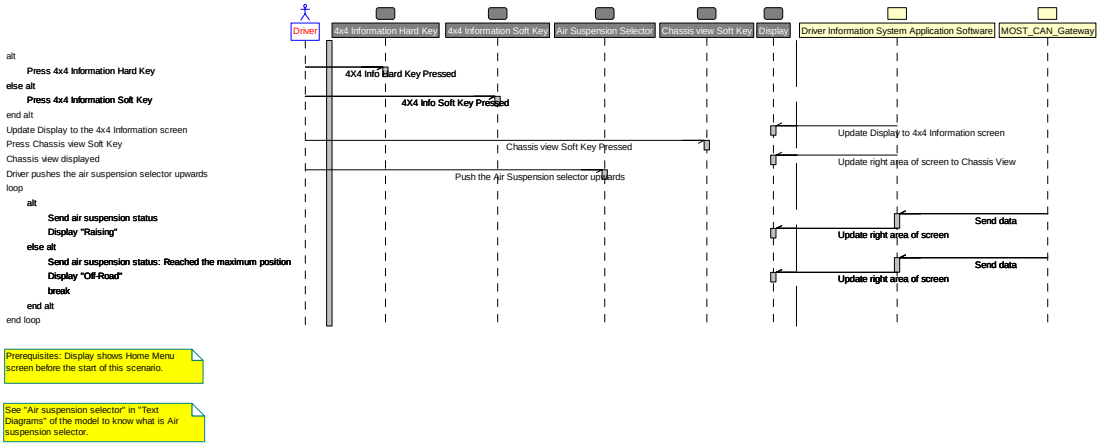


Fig. B-23. Sequence diagram 11: display of suspension status from home menu screen (HL - B).

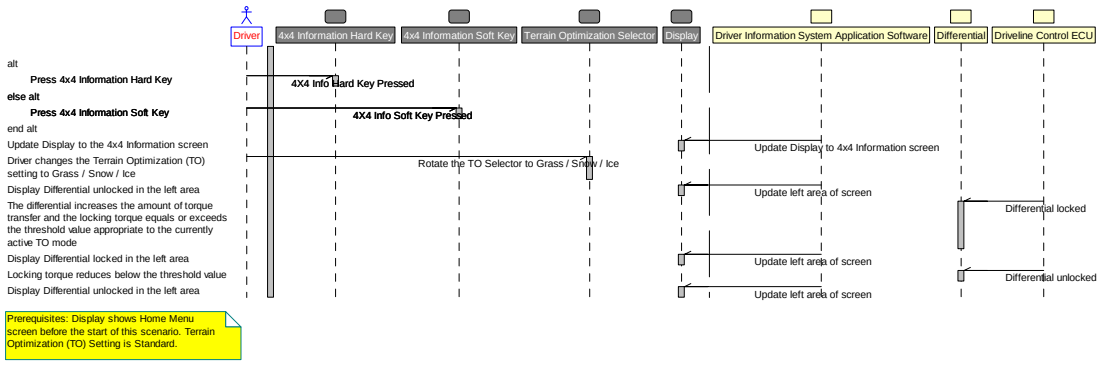


Fig. B-24. Sequence diagram 12: display of differential status from home menu screen (HL - B).

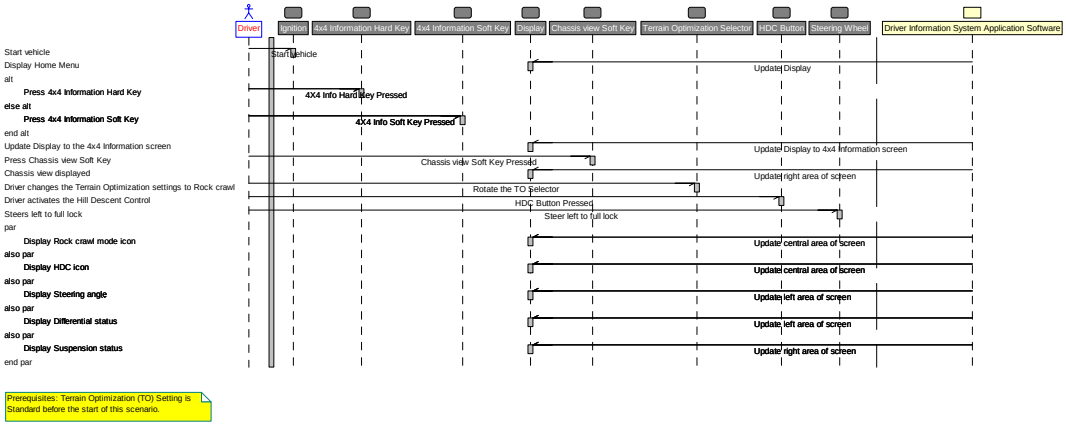


Fig. B-25. Sequence diagram 13: an off-road driving example (HL - B).

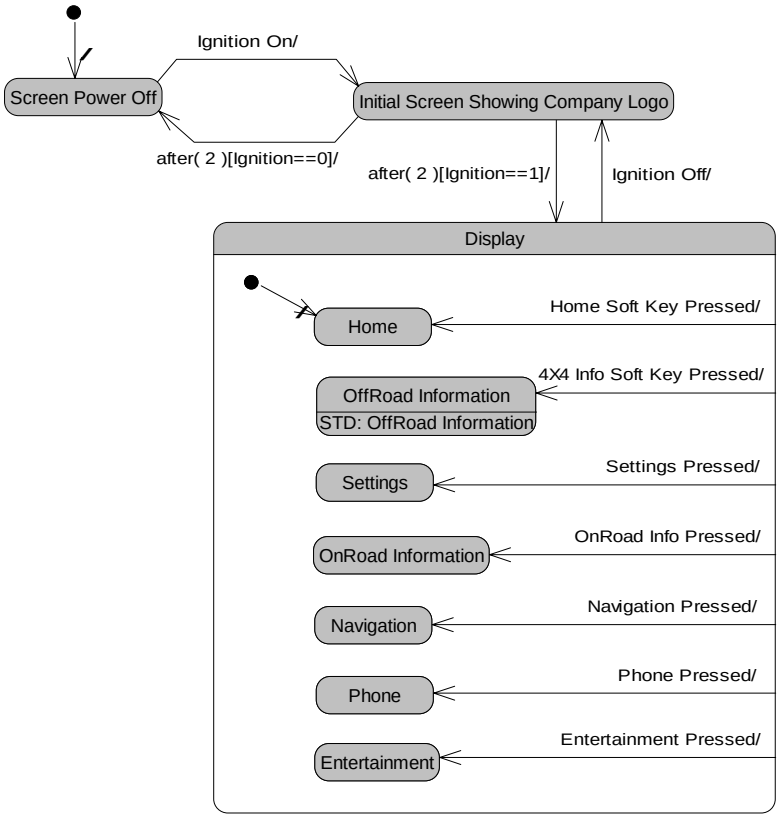


Fig. B-26. State machine diagram 1: Driver Information System.

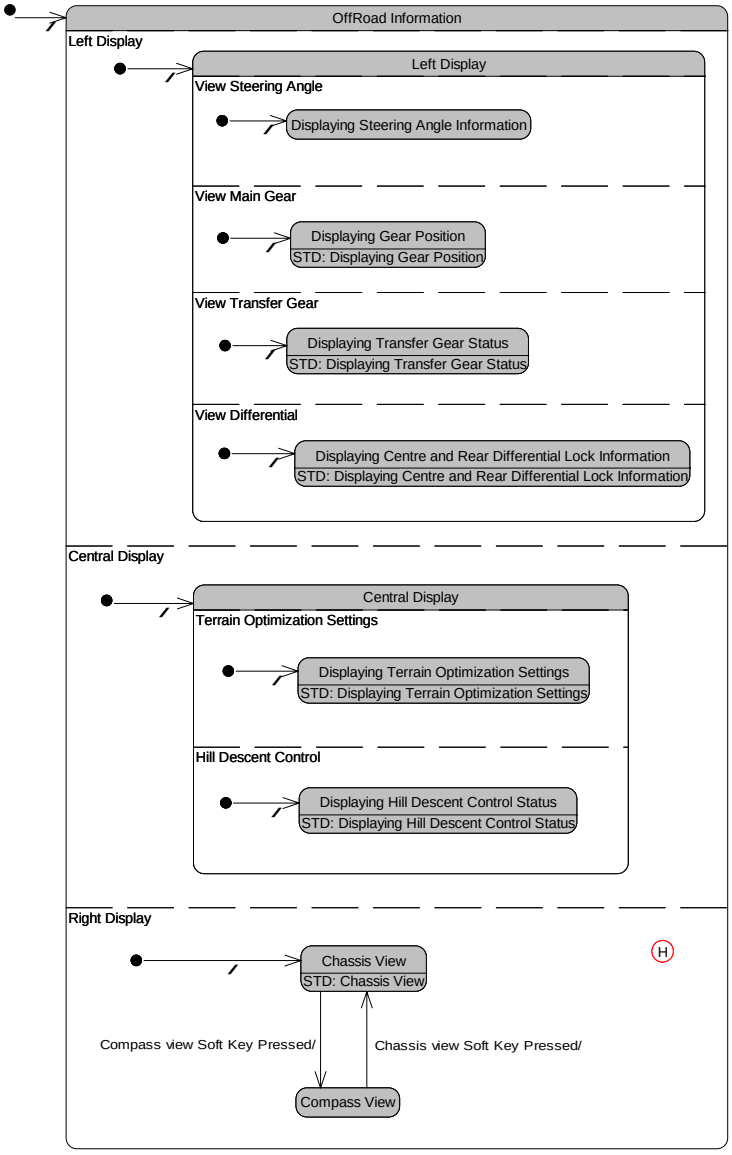


Fig. B-27. State machine diagram 2: off-road information.

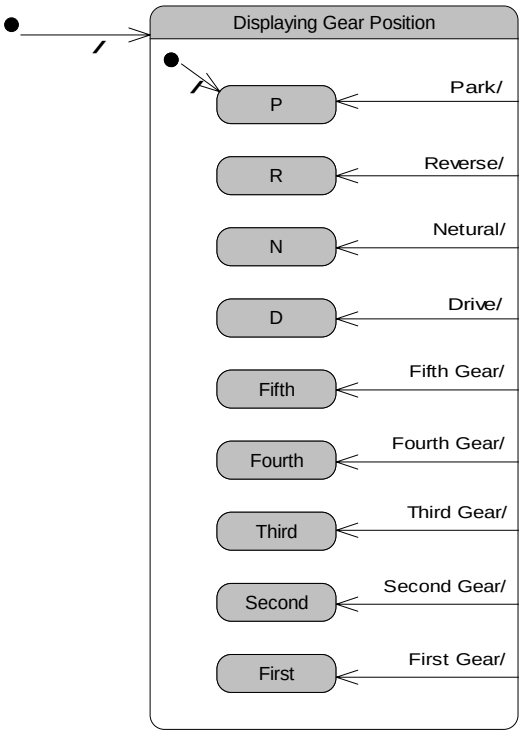


Fig. B-28. State machine diagram 3: displaying gear position.

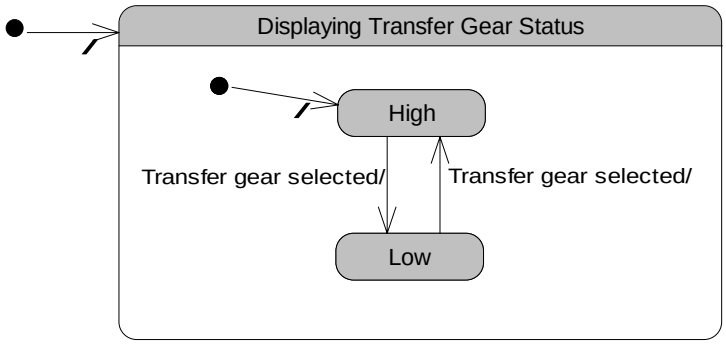


Fig. B-29. State machine diagram 4: displaying transfer gear status.

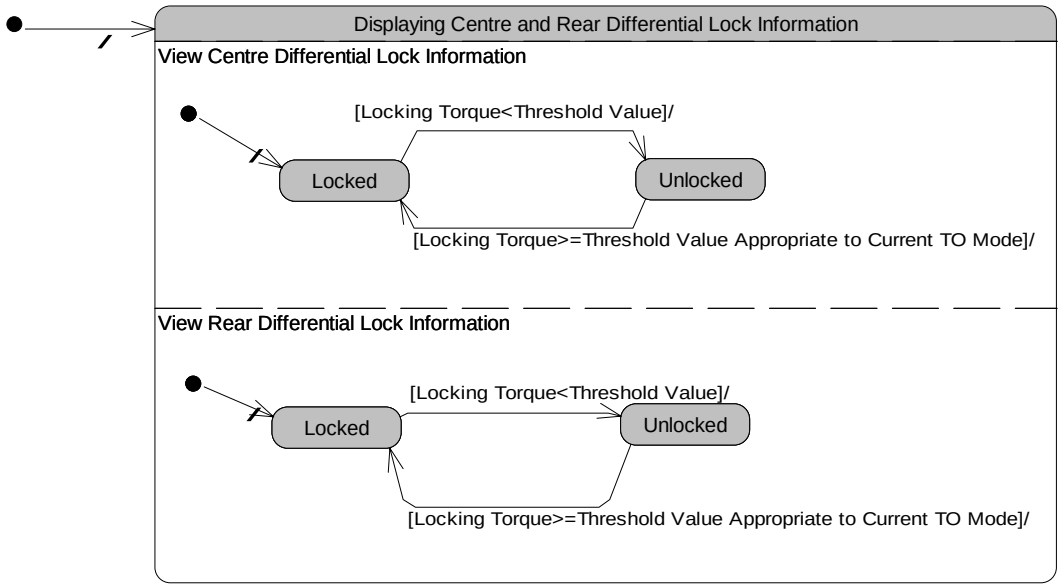


Fig. B-30. State machine diagram 5: displaying centre and rear differential lock information.

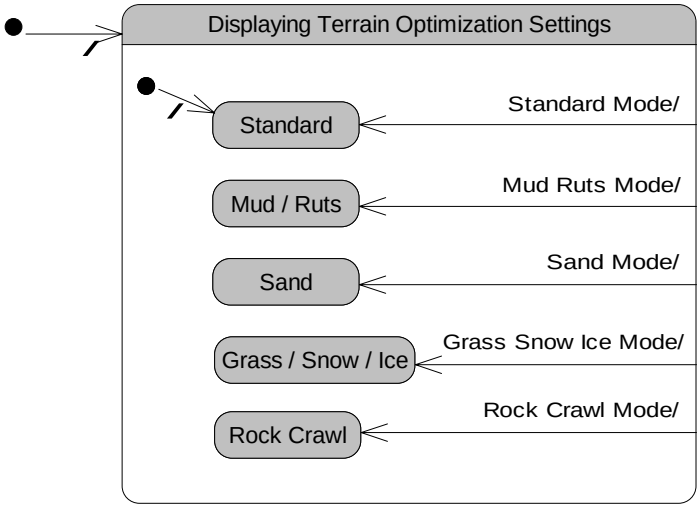


Fig. B-31. State machine diagram 6: displaying TO settings.

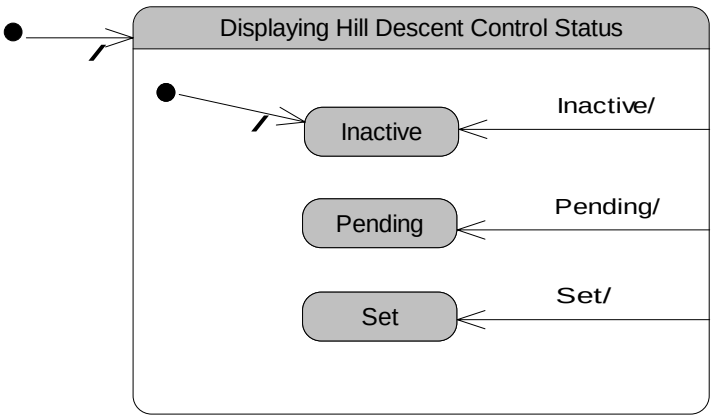


Fig. B-32. State machine diagram 7: displaying HDC status.

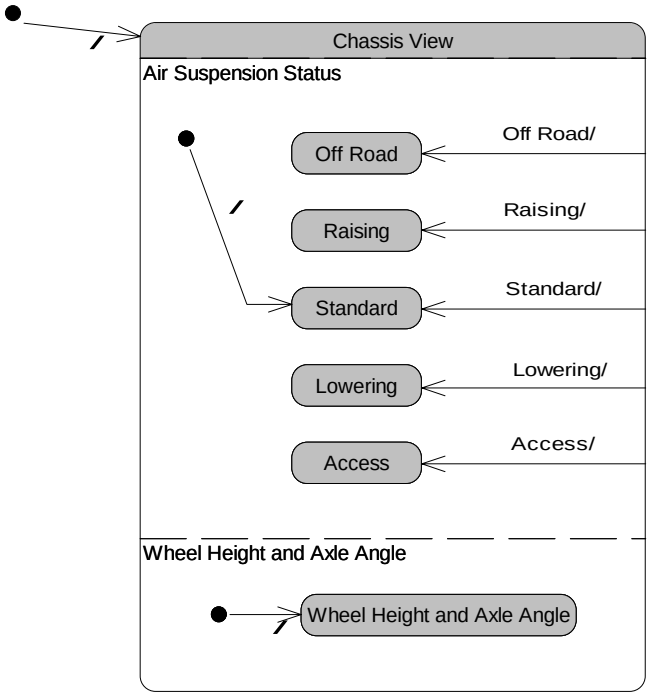


Fig. B-33. State machine diagram 8: chassis view.

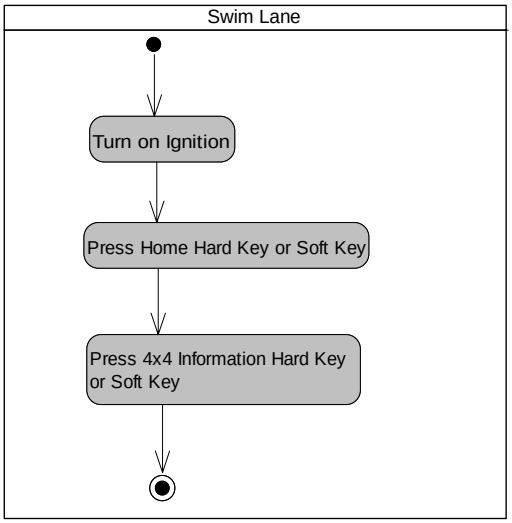


Fig. B-34. Activity diagram 1: getting to the 4×4 information screen.

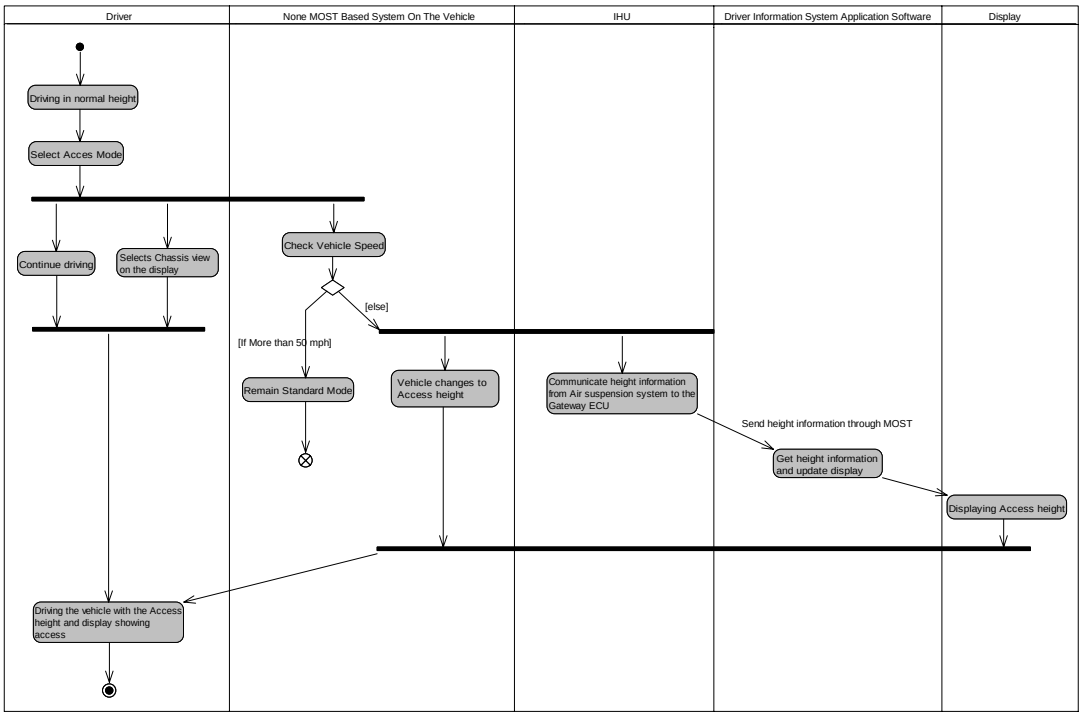


Fig. B-35. Activity diagram 2: select access height and viewing of new height information.

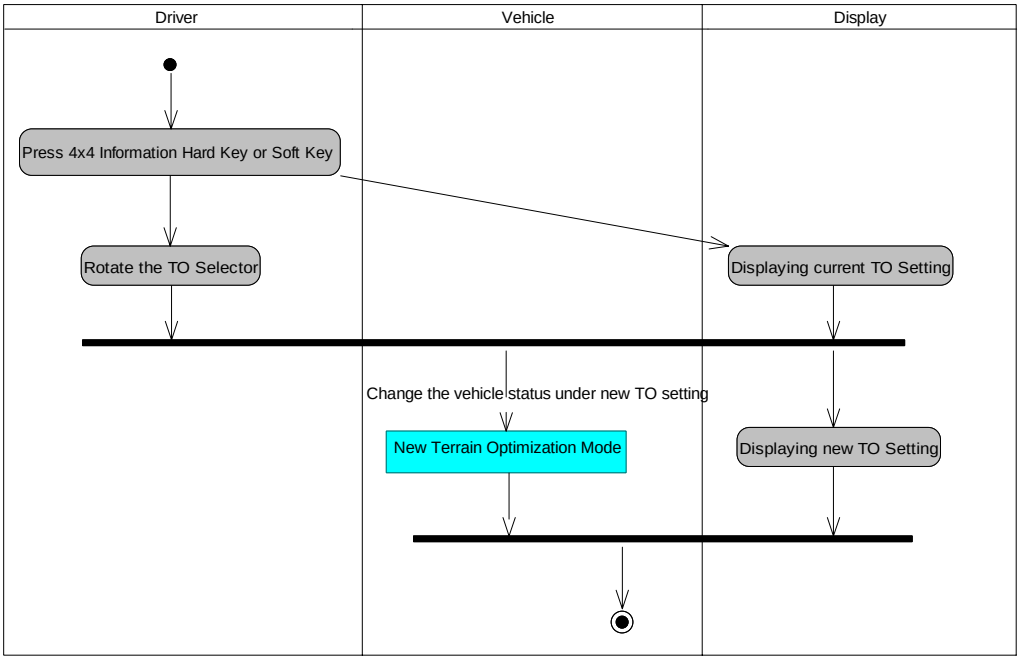


Fig. B-36. Activity diagram 3: view TO settings and change the TO mode.

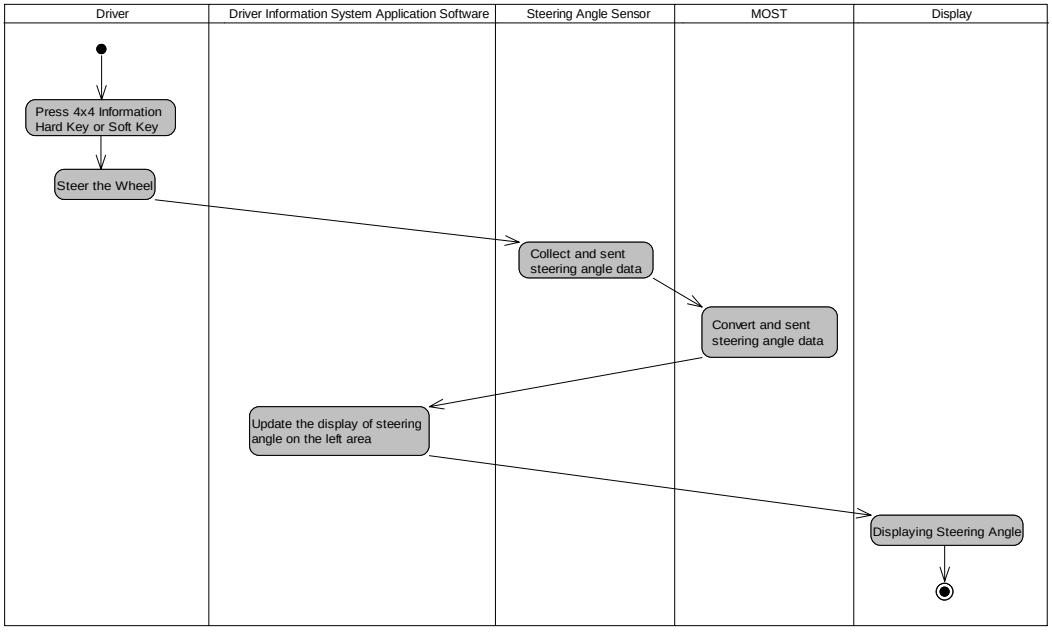


Fig. B-37. Activity diagram 4: view steering.

Appendix C

Diagrams in the model built in Simulink/Stateflow

This appendix provides a full list of Stateflow diagrams in the model that is developed in Simulink/Stateflow.

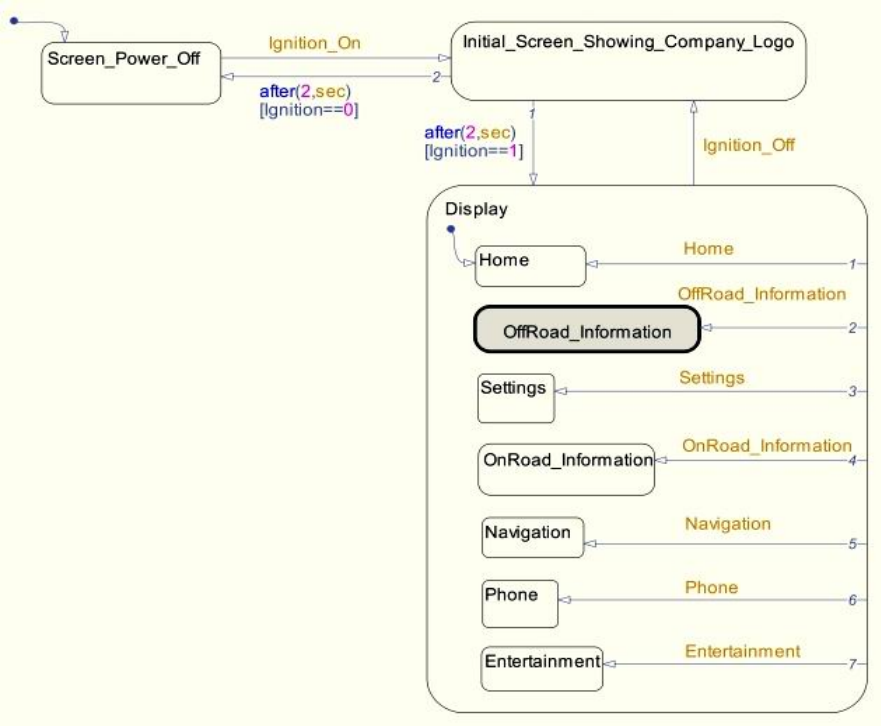


Fig. C-1. Stateflow diagram 1: Driver Information System.

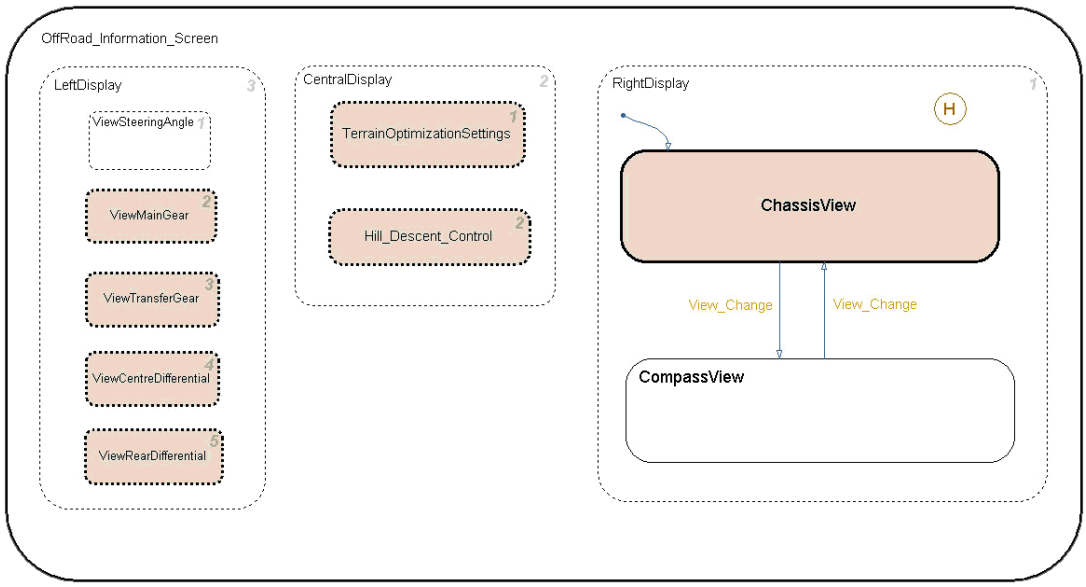


Fig. C-2. Stateflow diagram 2: display off-road information.

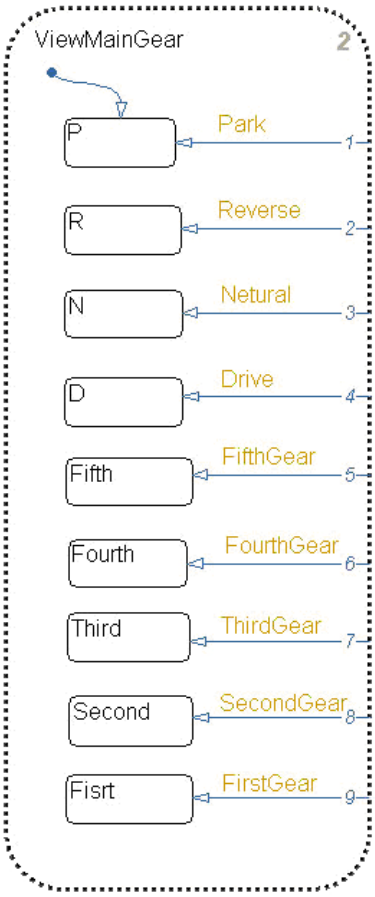


Fig. C-3. Stateflow diagram 3: view main gear.

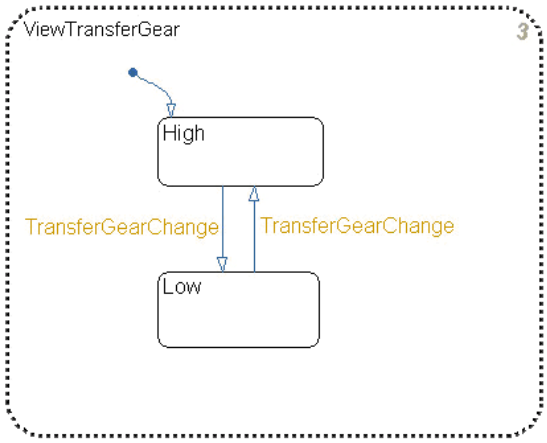


Fig. C-4. Stateflow diagram 4: view transfer gear.

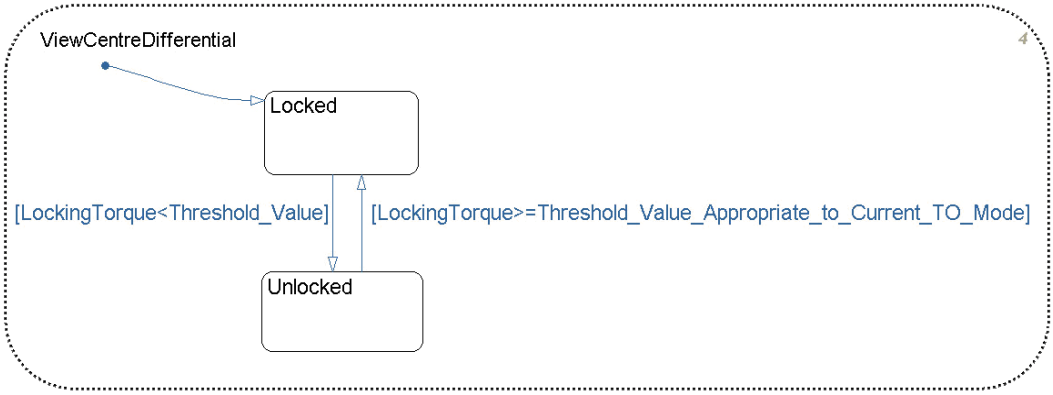


Fig. C-5. Stateflow diagram 5: view centre differential lock.

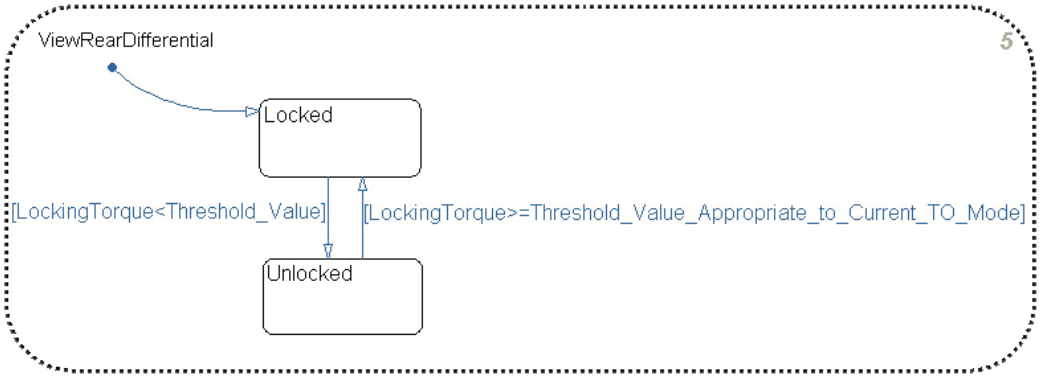


Fig. C-6. Stateflow diagram 6: view rear differential lock.

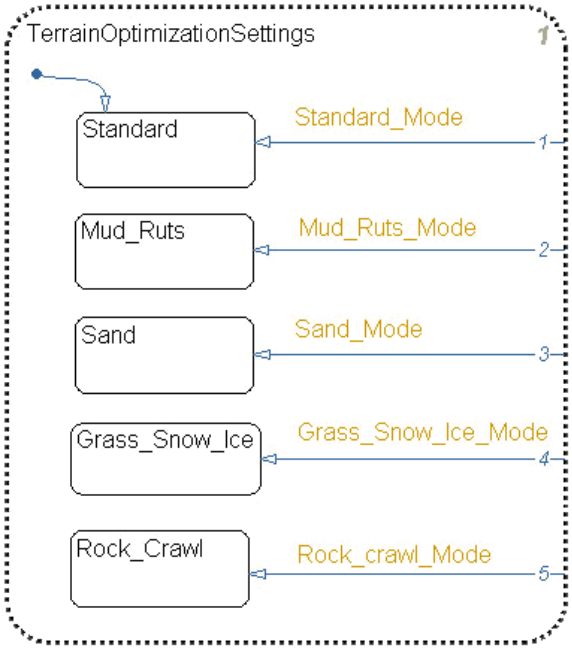


Fig. C-7. Stateflow diagram 7: view TO settings.

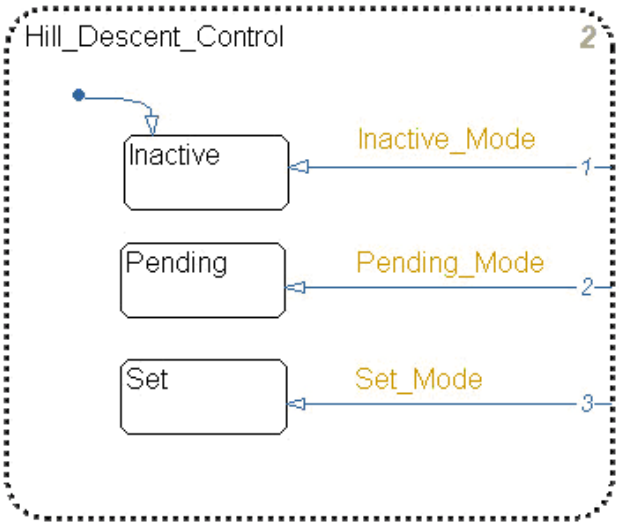


Fig. C-8. Stateflow diagram 8: view HDC status.

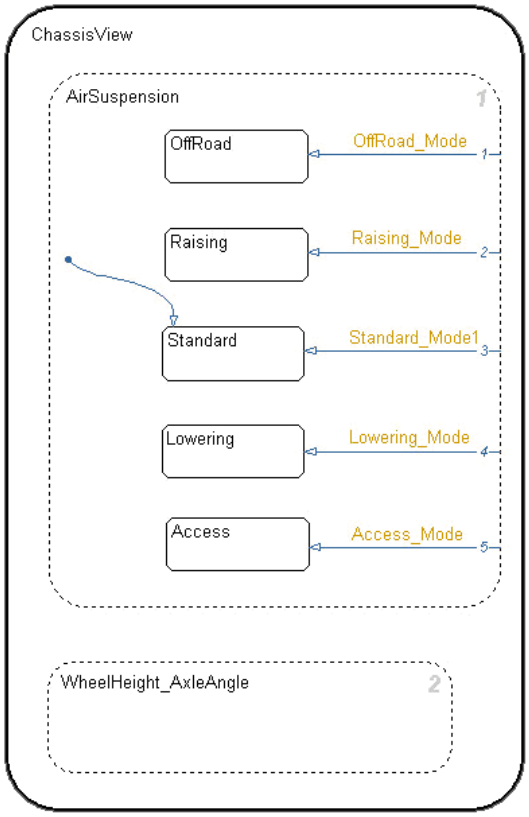


Fig. C-9. Stateflow diagram 9: chassis view.

Appendix D

Analysis result of the C code produced from ARTiSAN Studio

<polyspace-c C_R2008a>

Type C:\PolySpace_Results\kill-rte-kernel.bat on host ATA209 to halt Verifier process

Options used with Verifier:
-polyspace-version=C_R2008a
-date=09/02/2009
-main-generator-calls=unused
-lang=C
-results-dir=C:\PolySpace_Results
-author=admin-ata209
-main-generator-writes-variables=public
-target=sparc
-voa=true
-continue-with-red-error=true
-verif-version=1.0
-prog=New_Project
-D1=POLYSPACE_NO_STANDARD_STUBS
-D2=POLYSPACE_STRICT_ANSI_STANDARD_STUBS
-quick=true
-I1=E:\SysML Model C code for PolySpace
-I2=C:\Program Files\ARTiSAN Software Tools\ARTiSAN Real-time Studio\System\C_Sync
-desktop=true
-dos=true
-OS-target=no-predefined-OS

Verifying host configuration ...
Memory > 256MB : OK
(1015 MB)

```
Swap > 1GB :                                                                    OK
     (2.38 GB)
Swap >= 2*RAM :                                                                 OK
Tmp space available in C:\DOCUME~1\ADMIN-~1\LOCALS~1\Temp >= 10MB :     OK
     (824 MB)

*** Configuration of the host :      OK


Checking license ...
License is OK


PolySpace Technologies C static program verifier
Copyright 1999-2008, The MathWorks, Inc
All rights reserved.

Starting at: Feb 9, 2009 14:43:13
Host: MINGW32_XP-5.1 unknown 9 i686
User: admin-ata209
**********************************************************
***
*** Verifying C sources
***
**********************************************************
Copying sources to C-ALL ...

Number of files                 : 1
Number of lines                 : 4311
Number of lines without comments : 3142

OS-target no-predefined-OS implies: -D__STDC__

Verifying sources ...
Verifying Driver_Information_System.c

Verifying cross-files ANSI C compliance

Stubbing standard library functions ...
Stubbing unknown functions ...

Generating the Main ...
Generating call to function: RtsIgnition_On
Generating call to function: Rts4X4_Info_Soft_Key_Pressed
Generating call to function: RtsEntertainment_Pressed
Generating call to function: RtsHome_Soft_Key_Pressed
Generating call to function: RtsIgnition_Off
Generating call to function: RtsNavigation_Pressed
Generating call to function: RtsOnRoad_Info_Pressed
Generating call to function: RtsPhone_Pressed
Generating call to function: RtsSettings_Pressed
Generating call to function: RtsInactive
Generating call to function: RtsPending
Generating call to function: RtsSet
Generating call to function: RtsGrass_Snow_Ice_Mode
Generating call to function: RtsMud_Ruts_Mode
Generating call to function: RtsRock_Crawl_Mode
```

```
Generating call to function: RtsSand_Mode
Generating call to function: RtsStandard_Mode
Generating call to function: RtsDrive
Generating call to function: RtsFifth_Gear
Generating call to function: RtsFirst_Gear
Generating call to function: RtsFourth_Gear
Generating call to function: RtsNetural
Generating call to function: RtsPark
Generating call to function: RtsReverse
Generating call to function: RtsSecond_Gear
Generating call to function: RtsThird_Gear
Generating call to function: RtsTransfer_gear_selected
Generating call to function: RtsCompass_view_Soft_Key_Pressed
Generating call to function: RtsAccess
Generating call to function: RtsLowering
Generating call to function: RtsOff_Road
Generating call to function: RtsRaising
Generating call to function: RtsStandard
Generating call to function: RtsChassis_view_Soft_Key_Pressed
Generating call to function: Driver_Information_System
Generating call to function: _Driver_Information_System
Generating call to function: RtsInitial_Screen_Showing_Company_Logo_Timer1_CallBack
Generating call to function: RtsInitial_Screen_Showing_Company_Logo_Timer2_CallBack
Doing code transformations ...

**********************************************************
***
*** C sources verification done
***
**********************************************************
Ending at: Feb 9, 2009 14:44:17
User time for suif: 64.2real, 64.2u + 0s
Starting at: Feb 9, 2009 14:44:17
**********************************************************
***
*** Beginning C to intermediate language translation
***
**********************************************************
**** C to intermediate language translation 1 (P_SP)
**** C to intermediate language translation 1 (P_SP) took 2.9real, 2.9u + 0s
**** C to intermediate language translation 2 (P_RB)
**** C to intermediate language translation 2 (P_RB) took 0real, 0u + 0s
**** C to intermediate language translation 3 (P_SIA)
**** C to intermediate language translation 3 (P_SIA) took 2real, 2u + 0s
**** C to intermediate language translation 4 (P_CGA)
**** C to intermediate language translation 4 (P_CGA) took 2.1real, 2.1u + 0s
**** C to intermediate language translation 5 (P_SFNPV)
***** C to intermediate language translation 5.1 (P_PA)
****** C to intermediate language translation 5.1.1 (P_ATA)
****** C to intermediate language translation 5.1.1 (P_ATA) took 2real, 2u + 0s
****** C to intermediate language translation 5.1.2 (P_AP)
****** C to intermediate language translation 5.1.2 (P_AP) took 2.8real, 2.8u + 0s
****** C to intermediate language translation 5.1.3 (P_ITFP)
****** C to intermediate language translation 5.1.3 (P_ITFP) took 0.3real, 0.3u + 0s
****** C to intermediate language translation 5.1.4 (P_CA)
******* C to intermediate language translation 5.1.4.1 (P_STS)
******* C to intermediate language translation 5.1.4.1 (P_STS) took 0.4real, 0.4u + 0s
```

******* C to intermediate language translation 5.1.4.2 (P_RR)
******* C to intermediate language translation 5.1.4.2 (P_RR) took 0real, 0u + 0s
Some stats on aliases computation:
Number of aliases sets: 43
Number of couples of aliases: 77613
Number of elements in the biggest alias sets: 1st=279, 2nd=279, 3rd=4, 4th=4, 5th=2
****** C to intermediate language translation 5.1.4 (P_CA) took 0.5real, 0.5u + 0s
***** C to intermediate language translation 5.1 (P_PA) took 5.6real, 5.6u + 0s
***** C to intermediate language translation 5.2 (P_SSet)
***** C to intermediate language translation 5.2 (P_SSet) took 7.8real, 7.8u + 0s
***** C to intermediate language translation 5.3 (P_GA)
****** C to intermediate language translation 5.3.1 (P_FPGA)
****** C to intermediate language translation 5.3.1 (P_FPGA) took 2.1real, 2.1u + 0s
****** C to intermediate language translation 5.3.2 (P_GCPTS)
******* C to intermediate language translation 5.3.2.1 (P_GAA3)
******** C to intermediate language translation 5.3.2.1.1 (Loading)
******** C to intermediate language translation 5.3.2.1.1 (Loading) took 0real, 0u + 0s
[1121 -> 2655]
****** C to intermediate language translation 5.3.2 (P_GCPTS) took 1.4real, 1.4u + 0s
****** C to intermediate language translation 5.3.3 (P_MNPV)
****** C to intermediate language translation 5.3.3 (P_MNPV) took 2.6real, 2.6u + 0s
******** C to intermediate language translation 5.3.2.1.2 (P_GAA_SC)
********* C to intermediate language translation 5.3.2.1.2.1 (P_GAA_VI)
********* C to intermediate language translation 5.3.2.1.2.1 (P_GAA_VI) took 0real, 0u + 0s
********* C to intermediate language translation 5.3.2.1.2.2 (P_GAA_SDC)
********* C to intermediate language translation 5.3.2.1.2.2 (P_GAA_SDC) took 0real, 0u + 0s
********* C to intermediate language translation 5.3.2.1.2.3 (P_GAA_RS)
********* C to intermediate language translation 5.3.2.1.2.3 (P_GAA_RS) took 0real, 0u + 0s
******** C to intermediate language translation 5.3.2.1.2 (P_GAA_SC) took 0real, 0u + 0s
* inlining RtsRunToCompletion could decrease the number of aliases of parameter #1 from 94 to 6
* inlining RtsEnter_Screen_Power_Off could decrease the number of aliases of parameter #1 from 94 to 5
* inlining RtsExit_Compass_View could decrease the number of aliases of parameter #1 from 37 to 7
* inlining RtsExit_Chassis_View_1 could decrease the number of aliases of parameter #1 from 47 to 6
* inlining RtsExit_Chassis_View could decrease the number of aliases of parameter #1 from 37 to 6
* inlining RtsExit_Low could decrease the number of aliases of parameter #1 from 37 to 7
* inlining RtsExit_High could decrease the number of aliases of parameter #1 from 37 to 7
* inlining RtsExit_Displaying_Gear_Position_1 could decrease the number of aliases of parameter #1 from 53 to 5
* inlining RtsExit_Displaying_Terrain_Optimization_Settings_1 could decrease the number of aliases of parameter #1 from 45 to 6
* inlining RtsExit_Displaying_Hill_Descent_Control_Status_1 could decrease the number of aliases of parameter #1 from 41 to 5
******* C to intermediate language translation 5.3.2.1 (P_GAA3) took 15.4real, 15.4u + 0s
***** C to intermediate language translation 5.3 (P_GA) took 18.6real, 18.6u + 0s
***** C to intermediate language translation 5.4 (P_AA)
****** C to intermediate language translation 5.4.1 (P_AC)
Some stats on points to analysis:
Number of optimized point_to edges: 854
****** C to intermediate language translation 5.4.1 (P_AC) took 0real, 0u + 0s
***** C to intermediate language translation 5.4 (P_AA) took 0real, 0u + 0s
***** C to intermediate language translation 5.5 (P_PFF)

```
Found 1 polymorphic functions
***** C to intermediate language translation 5.5 (P_PFF) took 0.3real, 0.3u + 0s
***** C to intermediate language translation 5.6 (P_LGR)
***** C to intermediate language translation 5.6 (P_LGR) took 0real, 0u + 0s
**** C to intermediate language translation 5 (P_SFNPV) took 32.5real, 32.5u + 0s
**** C to intermediate language translation 6 (P_SP)
**** C to intermediate language translation 6 (P_SP) took 3.3real, 3.3u + 0s
**** C to intermediate language translation 7 (P_RB)
**** C to intermediate language translation 7 (P_RB) took 0real, 0u + 0s
**** C to intermediate language translation 8 (P_PA)
***** C to intermediate language translation 8.1 (P_ATA)
***** C to intermediate language translation 8.1 (P_ATA) took 2.2real, 2.2u + 0s
***** C to intermediate language translation 8.2 (P_AP)
***** C to intermediate language translation 8.2 (P_AP) took 3real, 3u + 0s
***** C to intermediate language translation 8.3 (P_ITFP)
***** C to intermediate language translation 8.3 (P_ITFP) took 0.2real, 0.2u + 0s
***** C to intermediate language translation 8.4 (P_CA)
****** C to intermediate language translation 8.4.1 (P_STS)
****** C to intermediate language translation 8.4.1 (P_STS) took 0.9real, 0.9u + 0s
****** C to intermediate language translation 8.4.2 (P_RR)
****** C to intermediate language translation 8.4.2 (P_RR) took 0.2real, 0.2u + 0s
Some stats on aliases computation:
   Number of aliases sets:          87
   Number of couples of aliases: 942013
   Number of elements in the biggest alias sets: 1st=316, 2nd=279, 3rd=279, 4th=279, 5th=279
***** C to intermediate language translation 8.4 (P_CA) took 1.3real, 1.3u + 0s
**** C to intermediate language translation 8 (P_PA) took 6.8real, 6.8u + 0s
**** C to intermediate language translation 9 (P_SSet)
**** C to intermediate language translation 9 (P_SSet) took 9.6real, 9.6u + 0s
**** C to intermediate language translation 10 (P_O)
**** C to intermediate language translation 10 (P_O) took 12.2real, 12.2u + 0s
**** C to intermediate language translation 11 (P_G)
**** C to intermediate language translation 11 (P_G) took 5.2real, 5.2u + 0s
**** C to intermediate language translation 12 (P_TT)
**** C to intermediate language translation 12 (P_TT) took 0.1real, 0.1u + 0s
**** C to intermediate language translation 13 (P_VT)
**** C to intermediate language translation 13 (P_VT) took 0real, 0u + 0s
**** C to intermediate language translation 14 (P_PT)
***** C to intermediate language translation 14.1 (P_SPP)
****** C to intermediate language translation 14.1.1 (P_CSSIP)
****** C to intermediate language translation 14.1.1 (P_CSSIP) took 0real, 0u + 0s
***** C to intermediate language translation 14.1 (P_SPP) took 0real, 0u + 0s
***** C to intermediate language translation 14.2 (P_TP)
 - translating procedure assert (1 / 235)
 - translating procedure RtsExit_Screen_Power_Off (2 / 235)
 - translating procedure RtsEnter_Initial_Screen_Showing_Company_Logo (3 / 235)
 - translating procedure RtsEnter_Screen_Power_Off (4 / 235)
 - translating procedure RtsRunToCompletion (5 / 235)
 - translating procedure RtsIgnition_On (6 / 235)
 - translating procedure RtsExit_Entertainment (7 / 235)
 - translating procedure RtsExit_Home (8 / 235)
 - translating procedure RtsExit_Navigation (9 / 235)
 - translating procedure RtsExit_Inactive (10 / 235)
 - translating procedure RtsExit_Pending (11 / 235)
 - translating procedure RtsExit_Set (12 / 235)
 - translating procedure RtsExit_Displaying_Hill_Descent_Control_Status_1 (13 / 235)
 - translating procedure RtsExit_Displaying_Hill_Descent_Control_Status (14 / 235)
```

- translating procedure RtsExit_Grass___Snow___Ice (15 / 235)
- translating procedure RtsExit_Mud___Ruts (16 / 235)
- translating procedure RtsExit_Rock_Crawl (17 / 235)
- translating procedure RtsExit_Sand (18 / 235)
- translating procedure RtsExit_Standard (19 / 235)
- translating procedure RtsExit_Displaying_Terrain_Optimization_Settings_1 (20 / 235)
- translating procedure RtsExit_Displaying_Terrain_Optimization_Settings (21 / 235)
- translating procedure RtsExit_Central_Display_1 (22 / 235)
- translating procedure RtsExit_Displaying_Centre_and_Rear_Differential_Lock_Information (23 / 235)
- translating procedure RtsExit_D (24 / 235)
- translating procedure RtsExit_Fifth (25 / 235)
- translating procedure RtsExit_First (26 / 235)
- translating procedure RtsExit_Fourth (27 / 235)
- translating procedure RtsExit_N (28 / 235)
- translating procedure RtsExit_P (29 / 235)
- translating procedure RtsExit_R (30 / 235)
- translating procedure RtsExit_Second (31 / 235)
- translating procedure RtsExit_Third (32 / 235)
- translating procedure RtsExit_Displaying_Gear_Position_1 (33 / 235)
- translating procedure RtsExit_Displaying_Gear_Position (34 / 235)
- translating procedure RtsExit_Displaying_Steering_Angle_Information (35 / 235)
- translating procedure RtsExit_High (36 / 235)
- translating procedure RtsExit_Low (37 / 235)
- translating procedure RtsExit_Displaying_Transfer_Gear_Status_1 (38 / 235)
- translating procedure RtsExit_Displaying_Transfer_Gear_Status (39 / 235)
- translating procedure RtsExit_Left_Display_1 (40 / 235)
- translating procedure RtsExit_Access (41 / 235)
- translating procedure RtsExit_Lowering (42 / 235)
- translating procedure RtsExit_Off_Road (43 / 235)
- translating procedure RtsExit_Raising (44 / 235)
- translating procedure RtsExit_Standard_1 (45 / 235)
- translating procedure RtsExit_Wheel_Height_and_Axle_Angle_1 (46 / 235)
- translating procedure RtsExit_Chassis_View_1 (47 / 235)
- translating procedure RtsExit_Chassis_View (48 / 235)
- translating procedure RtsExit_Compass_View (49 / 235)
- translating procedure RtsExit_OffRoad_Information_1 (50 / 235)
- translating procedure RtsExit_OffRoad_Information (51 / 235)
- translating procedure RtsExit_OnRoad_Information (52 / 235)
- translating procedure RtsExit_Phone (53 / 235)
- translating procedure RtsExit_Settings (54 / 235)
- translating procedure RtsExit_Display (55 / 235)
- translating procedure RtsEnter_OffRoad_Information (56 / 235)
- translating procedure RtsEnter_OffRoad_Information_1 (57 / 235)
- translating procedure RtsEnter_Central_Display_1 (58 / 235)
- translating procedure RtsEnter_Displaying_Hill_Descent_Control_Status (59 / 235)
- translating procedure RtsEnter_Displaying_Hill_Descent_Control_Status_1 (60 / 235)
- translating procedure RtsEnter_Inactive (61 / 235)
- translating procedure RtsDefault_Displaying_Hill_Descent_Control_Status_1 (62 / 235)
- translating procedure RtsDefault_Displaying_Hill_Descent_Control_Status (63 / 235)
- translating procedure RtsDefault_Hill_Descent_Control (64 / 235)
- translating procedure RtsEnter_Displaying_Terrain_Optimization_Settings (65 / 235)
- translating procedure RtsEnter_Displaying_Terrain_Optimization_Settings_1 (66 / 235)
- translating procedure RtsEnter_Standard (67 / 235)
- translating procedure RtsDefault_Displaying_Terrain_Optimization_Settings_1 (68 / 235)
- translating procedure RtsDefault_Displaying_Terrain_Optimization_Settings (69 / 235)
- translating procedure RtsDefault_Terrain_Optimization_Settings (70 / 235)

- translating procedure RtsDefault_Central_Display_1 (71 / 235)
- translating procedure RtsDefault_Central_Display (72 / 235)
- translating procedure RtsEnter_Left_Display_1 (73 / 235)
- translating procedure RtsEnter_Displaying_Centre_and_Rear_Differential_Lock_Information (74 / 235)
- translating procedure RtsDefault_View_Differential (75 / 235)
- translating procedure RtsEnter_Displaying_Gear_Position (76 / 235)
- translating procedure RtsEnter_Displaying_Gear_Position_1 (77 / 235)
- translating procedure RtsEnter_P (78 / 235)
- translating procedure RtsDefault_Displaying_Gear_Position_1 (79 / 235)
- translating procedure RtsDefault_Displaying_Gear_Position (80 / 235)
- translating procedure RtsDefault_View_Main_Gear (81 / 235)
- translating procedure RtsEnter_Displaying_Steering_Angle_Information (82 / 235)
- translating procedure RtsDefault_View_Steering_Angle (83 / 235)
- translating procedure RtsEnter_Displaying_Transfer_Gear_Status (84 / 235)
- translating procedure RtsEnter_Displaying_Transfer_Gear_Status_1 (85 / 235)
- translating procedure RtsEnter_High (86 / 235)
- translating procedure RtsDefault_Displaying_Transfer_Gear_Status_1 (87 / 235)
- translating procedure RtsDefault_Displaying_Transfer_Gear_Status (88 / 235)
- translating procedure RtsDefault_View_Transfer_Gear (89 / 235)
- translating procedure RtsDefault_Left_Display_1 (90 / 235)
- translating procedure RtsDefault_Left_Display (91 / 235)
- translating procedure RtsEnter_Chassis_View (92 / 235)
- translating procedure RtsEnter_Chassis_View_1 (93 / 235)
- translating procedure RtsEnter_Standard_1 (94 / 235)
- translating procedure RtsDefault_Air_Suspension_Status (95 / 235)
- translating procedure RtsEnter_Wheel_Height_and_Axle_Angle_1 (96 / 235)
- translating procedure RtsDefault_Wheel_Height_and_Axle_Angle (97 / 235)
- translating procedure RtsDefault_Chassis_View_1 (98 / 235)
- translating procedure RtsDefault_Chassis_View (99 / 235)
- translating procedure RtsDefault_Right_Display (100 / 235)
- translating procedure RtsDefault_OffRoad_Information_1 (101 / 235)
- translating procedure RtsDefault_OffRoad_Information (102 / 235)
- translating procedure Rts4X4_Info_Soft_Key_Pressed (103 / 235)
- translating procedure RtsEnter_Entertainment (104 / 235)
- translating procedure RtsEntertainment_Pressed (105 / 235)
- translating procedure RtsEnter_Home (106 / 235)
- translating procedure RtsHome_Soft_Key_Pressed (107 / 235)
- translating procedure RtsIgnition_Off (108 / 235)
- translating procedure RtsEnter_Navigation (109 / 235)
- translating procedure RtsNavigation_Pressed (110 / 235)
- translating procedure RtsEnter_OnRoad_Information (111 / 235)
- translating procedure RtsOnRoad_Info_Pressed (112 / 235)
- translating procedure RtsEnter_Phone (113 / 235)
- translating procedure RtsPhone_Pressed (114 / 235)
- translating procedure RtsEnter_Settings (115 / 235)
- translating procedure RtsSettings_Pressed (116 / 235)
- translating procedure RtsInactive (117 / 235)
- translating procedure RtsEnter_Pending (118 / 235)
- translating procedure RtsPending (119 / 235)
- translating procedure RtsEnter_Set (120 / 235)
- translating procedure RtsSet (121 / 235)
- translating procedure RtsEnter_Grass___Snow___Ice (122 / 235)
- translating procedure RtsGrass_Snow_Ice_Mode (123 / 235)
- translating procedure RtsEnter_Mud___Ruts (124 / 235)
- translating procedure RtsMud_Ruts_Mode (125 / 235)
- translating procedure RtsEnter_Rock_Crawl (126 / 235)

- translating procedure RtsRock_Crawl_Mode (127 / 235)
- translating procedure RtsEnter_Sand (128 / 235)
- translating procedure RtsSand_Mode (129 / 235)
- translating procedure RtsStandard_Mode (130 / 235)
- translating procedure RtsEnter_D (131 / 235)
- translating procedure RtsDrive (132 / 235)
- translating procedure RtsEnter_Fifth (133 / 235)
- translating procedure RtsFifth_Gear (134 / 235)
- translating procedure RtsEnter_First (135 / 235)
- translating procedure RtsFirst_Gear (136 / 235)
- translating procedure RtsEnter_Fourth (137 / 235)
- translating procedure RtsFourth_Gear (138 / 235)
- translating procedure RtsEnter_N (139 / 235)
- translating procedure RtsNetural (140 / 235)
- translating procedure RtsPark (141 / 235)
- translating procedure RtsEnter_R (142 / 235)
- translating procedure RtsReverse (143 / 235)
- translating procedure RtsEnter_Second (144 / 235)
- translating procedure RtsSecond_Gear (145 / 235)
- translating procedure RtsEnter_Third (146 / 235)
- translating procedure RtsThird_Gear (147 / 235)
- translating procedure RtsEnter_Low (148 / 235)
- translating procedure RtsTransfer_gear_selected (149 / 235)
- translating procedure RtsEnter_Compass_View (150 / 235)
- translating procedure RtsCompass_view_Soft_Key_Pressed (151 / 235)
- translating procedure RtsEnter_Access (152 / 235)
- translating procedure RtsAccess (153 / 235)
- translating procedure RtsEnter_Lowering (154 / 235)
- translating procedure RtsLowering (155 / 235)
- translating procedure RtsEnter_Off_Road (156 / 235)
- translating procedure RtsOff_Road (157 / 235)
- translating procedure RtsEnter_Raising (158 / 235)
- translating procedure RtsRaising (159 / 235)
- translating procedure RtsStandard (160 / 235)
- translating procedure RtsChassis_view_Soft_Key_Pressed (161 / 235)
- translating procedure Driver_Information_System (162 / 235)
- translating procedure _Driver_Information_System (163 / 235)
- translating procedure RtsExit_Initial_Screen_Showing_Company_Logo (164 / 235)
- translating procedure RtsEnter_Display (165 / 235)
- translating procedure RtsDefault_Display (166 / 235)
- translating procedure RtsInitial_Screen_Showing_Company_Logo_Timer1 (167 / 235)
- translating procedure RtsInitial_Screen_Showing_Company_Logo_Timer1_CallBack (168 / 235)
- translating procedure RtsInitial_Screen_Showing_Company_Logo_Timer2 (169 / 235)
- translating procedure RtsInitial_Screen_Showing_Company_Logo_Timer2_CallBack (170 / 235)
- translating procedure _main_gen_init_g18 (171 / 235)
- translating procedure _main_gen_init_g19 (172 / 235)
- translating procedure _main_gen_init_g20 (173 / 235)
- translating procedure _main_gen_init_g21 (174 / 235)
- translating procedure _main_gen_init_g22 (175 / 235)
- translating procedure _main_gen_init_g23 (176 / 235)
- translating procedure _main_gen_init_g24 (177 / 235)
- translating procedure _main_gen_init_g25 (178 / 235)
- translating procedure _main_gen_init_g26 (179 / 235)
- translating procedure _main_gen_init_g27 (180 / 235)
- translating procedure _main_gen_init_g28 (181 / 235)

```
 - translating procedure _main_gen_init_g29 (182 / 235)
 - translating procedure _main_gen_init_g30 (183 / 235)
 - translating procedure _main_gen_init_g31 (184 / 235)
 - translating procedure _main_gen_init_g32 (185 / 235)
 - translating procedure _main_gen_init_g33 (186 / 235)
 - translating procedure _main_gen_init_g34 (187 / 235)
 - translating procedure _main_gen_init_g35 (188 / 235)
 - translating procedure _main_gen_init_g36 (189 / 235)
 - translating procedure _main_gen_init_g37 (190 / 235)
 - translating procedure _main_gen_init_g38 (191 / 235)
 - translating procedure _main_gen_init_g39 (192 / 235)
 - translating procedure _main_gen_init_g40 (193 / 235)
 - translating procedure _main_gen_init_g17 (194 / 235)
 - translating procedure _main_gen_call_RtsIgnition_On (195 / 235)
 - translating procedure _main_gen_call_Rts4X4_Info_Soft_Key_Pressed (196 / 235)
 - translating procedure _main_gen_call_RtsEntertainment_Pressed (197 / 235)
 - translating procedure _main_gen_call_RtsHome_Soft_Key_Pressed (198 / 235)
 - translating procedure _main_gen_call_RtsIgnition_Off (199 / 235)
 - translating procedure _main_gen_call_RtsNavigation_Pressed (200 / 235)
 - translating procedure _main_gen_call_RtsOnRoad_Info_Pressed (201 / 235)
 - translating procedure _main_gen_call_RtsPhone_Pressed (202 / 235)
 - translating procedure _main_gen_call_RtsSettings_Pressed (203 / 235)
 - translating procedure _main_gen_call_RtsInactive (204 / 235)
 - translating procedure _main_gen_call_RtsPending (205 / 235)
 - translating procedure _main_gen_call_RtsSet (206 / 235)
 - translating procedure _main_gen_call_RtsGrass_Snow_Ice_Mode (207 / 235)
 - translating procedure _main_gen_call_RtsMud_Ruts_Mode (208 / 235)
 - translating procedure _main_gen_call_RtsRock_Crawl_Mode (209 / 235)
 - translating procedure _main_gen_call_RtsSand_Mode (210 / 235)
 - translating procedure _main_gen_call_RtsStandard_Mode (211 / 235)
 - translating procedure _main_gen_call_RtsDrive (212 / 235)
 - translating procedure _main_gen_call_RtsFifth_Gear (213 / 235)
 - translating procedure _main_gen_call_RtsFirst_Gear (214 / 235)
 - translating procedure _main_gen_call_RtsFourth_Gear (215 / 235)
 - translating procedure _main_gen_call_RtsNetural (216 / 235)
 - translating procedure _main_gen_call_RtsPark (217 / 235)
 - translating procedure _main_gen_call_RtsReverse (218 / 235)
 - translating procedure _main_gen_call_RtsSecond_Gear (219 / 235)
 - translating procedure _main_gen_call_RtsThird_Gear (220 / 235)
 - translating procedure _main_gen_call_RtsTransfer_gear_selected (221 / 235)
 - translating procedure _main_gen_call_RtsCompass_view_Soft_Key_Pressed (222 / 235)
 - translating procedure _main_gen_call_RtsAccess (223 / 235)
 - translating procedure _main_gen_call_RtsLowering (224 / 235)
 - translating procedure _main_gen_call_RtsOff_Road (225 / 235)
 - translating procedure _main_gen_call_RtsRaising (226 / 235)
 - translating procedure _main_gen_call_RtsStandard (227 / 235)
 - translating procedure _main_gen_call_RtsChassis_view_Soft_Key_Pressed (228 / 235)
 - translating procedure _main_gen_call_Driver_Information_System (229 / 235)
 - translating procedure _main_gen_call__Driver_Information_System (230 / 235)
 - translating procedure _main_gen_call_RtsInitial_Screen_Showing_Company_Logo_Timer1_CallBack (231 / 235)
 - translating procedure _main_gen_call_RtsInitial_Screen_Showing_Company_Logo_Timer2_CallBack (232 / 235)
 - translating procedure main (233 / 235)
 - translating procedure __PST__MAIN__ENTRY__POINT__ (234 / 235)
Some stats on aliases use:
   Number of alias writes:          2823
```

```
   Number of must-alias writes: 364
   Number of pma writes:        364
   Number of alias reads:       0
   Number of invisibles:        376
Stats about alias writes:
   biggest sets of alias writes: RtsEnter_Screen_Power_Off:this (77),
RtsExit_Displaying_Gear_Position_1:this (39),
                                 RtsExit_Chassis_View_1:this (32)
   procedures that write the biggest sets of aliases: RtsExit_Left_Display_1 (88),
RtsEnter_Screen_Power_Off (77),
                                                    RtsExit_Chassis_View_1
(64)
***** C to intermediate language translation 14.2 (P_TP) took 39.1real, 39.1u + 0s
**** C to intermediate language translation 14 (P_PT) took 39.2real, 39.2u + 0s
**** C to intermediate language translation 15 (P_IL)
***** C to intermediate language translation 15.1 (P_DRP)
***** C to intermediate language translation 15.1 (P_DRP) took 0real, 0u + 0s
***** C to intermediate language translation 15.2 (P_DR)
***** C to intermediate language translation 15.2 (P_DR) took 0real, 0u + 0s
***** C to intermediate language translation 15.3 (P_IGA)
***** C to intermediate language translation 15.3 (P_IGA) took 7.1real, 7.1u + 0s
***** C to intermediate language translation 15.4 (P_AG)
0 constructions broken due to gotos
***** C to intermediate language translation 15.4 (P_AG) took 6.2real, 6.2u + 0s
***** C to intermediate language translation 15.5 (P_CG)
***** C to intermediate language translation 15.5 (P_CG) took 4.1real, 4.1u + 0s
***** C to intermediate language translation 15.6 (P_R)
***** C to intermediate language translation 15.6 (P_R) took 4.6real, 4.6u + 0s
***** C to intermediate language translation 15.7 (P_PP)
* 362 pp, 420 ppp.
***** C to intermediate language translation 15.7 (P_PP) took 4.7real, 4.7u + 0s
***** C to intermediate language translation 15.8 (P_ICSP)
* 4167 cd, 11752 cf, 0 rc, 0 ff, 0 ed, 0 cd.
***** C to intermediate language translation 15.8 (P_ICSP) took 25.8real, 25.8u + 0s
***** C to intermediate language translation 15.9 (P_ILA)
***** C to intermediate language translation 15.9 (P_ILA) took 6.3real, 6.3u + 0s
***** C to intermediate language translation 15.10 (P_PGC)
* 4461 tdl.
***** C to intermediate language translation 15.10 (P_PGC) took 15.8real, 15.8u + 0s
***** C to intermediate language translation 15.11 (P_ILA)
***** C to intermediate language translation 15.11 (P_ILA) took 5.6real, 5.6u + 0s
***** C to intermediate language translation 15.12 (P_PGC)
* 2797 tdl.
***** C to intermediate language translation 15.12 (P_PGC) took 13.5real, 13.5u + 0s
***** C to intermediate language translation 15.13 (P_SULV)
***** C to intermediate language translation 15.13 (P_SULV) took 3.8real, 3.8u + 0s
***** C to intermediate language translation 15.14 (P_ICPP)
***** C to intermediate language translation 15.14 (P_ICPP) took 18real, 18u + 0s
***** C to intermediate language translation 15.15 (P_PP)
* 0 pp, 0 ppp.
***** C to intermediate language translation 15.15 (P_PP) took 5.9real, 5.9u + 0s
***** C to intermediate language translation 15.16 (P_SRC)
* 105 rcd, 0 tpd.
***** C to intermediate language translation 15.16 (P_SRC) took 11.2real, 11.2u + 0s
***** C to intermediate language translation 15.17 (P_SULV)
***** C to intermediate language translation 15.17 (P_SULV) took 4.1real, 4.1u + 0s
***** C to intermediate language translation 15.18 (P_SENUP)
```

```
3 empty procedure(s) removed
***** C to intermediate language translation 15.18 (P_SENUP) took 0.1real, 0.1u + 0s
***** C to intermediate language translation 15.19 (P_R)
***** C to intermediate language translation 15.19 (P_R) took 3.1real, 3.1u + 0s
**** C to intermediate language translation 15 (P_IL) took 216.7real, 216.7u + 0s
0 empty package(s) removed
**** C to intermediate language translation 16 (P_IPF)
94% init procedures removed
**** C to intermediate language translation 16 (P_IPF) took 3real, 3u + 0s
74% types removed
* assigns: 52% reduction
* asserts: 48% reduction
* total    : 55% reduction
************************************************************
***
*** C to intermediate language translation done
***
************************************************************
Ending at: Feb 9, 2009 14:50:34
User time for iabc-c2if: 376.5real, 376.5u + 0s
Starting at: Feb 9, 2009 14:50:34
************************************************************
***
*** Beginning Quick Software Safety Integration Analysis
***
************************************************************
**** Quick Software Safety Integration Analysis 1 (MF)
**** Quick Software Safety Integration Analysis 1 (MF) took 3.4real, 3.4u + 0s
**** Quick Software Safety Integration Analysis 2 (interprocedural propagation)
**** Quick Software Safety Integration Analysis 2 (interprocedural propagation) took 16.3real,
16.3u + 0s

Generating GUI files
Checks statistics: (including internal files)
  - IRV       => Green :       23, Orange :        0, Red :        0, Gray :        0     (100%)
  - OVFL      => Green :       38, Orange :        0, Red :        0, Gray :        0     (100%)
  - NIP       => Green :      537, Orange :        0, Red :        0, Gray :      131     (100%)
  - NIVL      => Green :      215, Orange :        0, Red :        0, Gray :        0     (100%)
  - NIV       => Green :      100, Orange :      193, Red :        0, Gray :      131     (54%)
  - UNFL      => Green :       38, Orange :        0, Red :        0, Gray :        0     (100%)
  - COR       => Green :      152, Orange :        0, Red :        0, Gray :        0     (100%)
  - OBAI      => Green :       38, Orange :        0, Red :        0, Gray :        0     (100%)
  - ZDV       => Green :      152, Orange :        0, Red :        0, Gray :        0     (100%)
  - IDP       => Green :      205, Orange :       84, Red :        0, Gray :      131     (80%)

TOTAL:        => Green :     1498, Orange :      277, Red :        0, Gray :      393     (87%)

Number of NTL : 0
Number of NTC : 0
Number of UNR : 0


GUI files generation complete.

************************************************************
***
*** Quick Software Safety Integration Analysis done
```

***
****
Ending at: Feb 9, 2009 14:51:14
User time for quick: 40.1real, 40.1u + 0s
User time for polyspace-c: 482.4real, 482.4u + 0s

***
*** End of PolySpace Verifier analysis
***

Appendix E

Analysis result of the C code produced from Real-Time Workshop

<polyspace-c C_R2008a>

Type C:\PolySpace_Results\kill-rte-kernel.bat on host ATA209 to halt Verifier process

Options used with Verifier:
-polyspace-version=C_R2008a
-date=09/02/2009
-main-generator-calls=unused
-lang=C
-results-dir=C:\PolySpace_Results
-author=admin-ata209
-main-generator-writes-variables=public
-target=sparc
-voa=true
-continue-with-red-error=true
-verif-version=1.0
-prog=New_Project
-D1=POLYSPACE_NO_STANDARD_STUBS
-quick=true
-I1=E:\FunctionalModel0209_ert_rtw
-I2=C:\MATLAB704\sys\lcc\include
-I3=C:\MATLAB704\simulink\include
-I4=C:\MATLAB704\rtw\c\libsrc
-I5=C:\MATLAB704\extern\include
-desktop=true
-dos=true
-OS-target=no-predefined-OS

Verifying host configuration ...

```
Memory > 256MB :                                                    OK
      (1015 MB)
Swap > 1GB :                                                        OK
      (2.38 GB)
Swap >= 2*RAM :                                                     OK
Tmp space available in C:\DOCUME~1\ADMIN-~1\LOCALS~1\Temp >= 10MB :    OK
      (823 MB)

*** Configuration of the host :      OK


Checking license ...
License is OK


PolySpace Technologies C static program verifier
Copyright 1999-2008, The MathWorks, Inc
All rights reserved.

Starting at: Feb 9, 2009 14:56:23
Host: MINGW32_XP-5.1 unknown 9 i686
User: admin-ata209
**********************************************************
***
*** Verifying C sources
***
**********************************************************
Copying sources to C-ALL ...

Number of files                 : 3
Number of lines                 : 1818
Number of lines without comments : 1336

OS-target no-predefined-OS implies: -D__STDC__

Verifying sources ...
Verifying FunctionalModel0209.c
Verifying FunctionalModel0209_data.c
Verifying ert_main.c

Verifying cross-files ANSI C compliance

Stubbing standard library functions ...
Stubbing unknown functions ...
* Function fflush may write to its arguments and may return random.
            Does not model pointer effects. Returns an initialized value.
* Function memset may write to its arguments and may return random.
            Does not model pointer effects. Returns an initialized value.
            Const parameters (nb params=3): (#2, #3).
* Function rt_ZCFcn may write to its arguments and may return random.
            Does not model pointer effects. Returns an initialized value.
            Const parameters (nb params=3): (#1, #3).
* Function printf may write to its arguments and may return random.
            Does not model pointer effects.
            It may write in the variable arguments. Returns an initialized value.
            Const parameters (nb params=3): #1.
* Function floor is pure. Returns an initialized value.
```

```
                Const parameters (nb params=1): #1.

Generating the Main ...

Warning: a main procedure already exists.
                No main will be generated: the existing one will be used...
Doing code transformations ...

************************************************************
***
*** C sources verification done
***
************************************************************
Ending at: Feb 9, 2009 14:57:32
User time for suif: 69.3real, 69.3u + 0s
Starting at: Feb 9, 2009 14:57:32
************************************************************
***
*** Beginning C to intermediate language translation
***
************************************************************
**** C to intermediate language translation 1 (P_SP)
**** C to intermediate language translation 1 (P_SP) took 0.9real, 0.9u + 0s
**** C to intermediate language translation 2 (P_RB)
**** C to intermediate language translation 2 (P_RB) took 0real, 0u + 0s
rt_OneStep is dead code
FunctionalModel0209_step is dead code
Functio_DriverInformationSystem is dead code
chartstep_c1_FunctionalModel020 is dead code
FunctionalModel0209_Display is dead code
Fun_enter_internal_RightDisplay is dead code
Functiona_exit_internal_Display is dead code
rt_ZCFcn is dead code
**** C to intermediate language translation 3 (P_SIA)
**** C to intermediate language translation 3 (P_SIA) took 0.3real, 0.3u + 0s
**** C to intermediate language translation 4 (P_CGA)
**** C to intermediate language translation 4 (P_CGA) took 0.1real, 0.1u + 0s
**** C to intermediate language translation 5 (P_SFNPV)
***** C to intermediate language translation 5.1 (P_PA)
****** C to intermediate language translation 5.1.1 (P_ATA)
****** C to intermediate language translation 5.1.1 (P_ATA) took 0.2real, 0.2u + 0s
****** C to intermediate language translation 5.1.2 (P_AP)
****** C to intermediate language translation 5.1.2 (P_AP) took 0.1real, 0.1u + 0s
****** C to intermediate language translation 5.1.3 (P_ITFP)
****** C to intermediate language translation 5.1.3 (P_ITFP) took 0real, 0u + 0s
****** C to intermediate language translation 5.1.4 (P_CA)
******* C to intermediate language translation 5.1.4.1 (P_STS)
******* C to intermediate language translation 5.1.4.1 (P_STS) took 0.1real, 0.1u + 0s
******* C to intermediate language translation 5.1.4.2 (P_RR)
******* C to intermediate language translation 5.1.4.2 (P_RR) took 0real, 0u + 0s
Some stats on aliases computation:
   Number of aliases sets:          11
   Number of couples of aliases: 45
   Number of elements in the biggest alias sets: 1st=8, 2nd=3, 3rd=3, 4th=3, 5th=3
****** C to intermediate language translation 5.1.4 (P_CA) took 0.1real, 0.1u + 0s
***** C to intermediate language translation 5.1 (P_PA) took 0.5real, 0.5u + 0s
***** C to intermediate language translation 5.2 (P_SSet)
```

```
***** C to intermediate language translation 5.2 (P_SSet) took 0.8real, 0.8u + 0s
***** C to intermediate language translation 5.3 (P_GA)
****** C to intermediate language translation 5.3.1 (P_FPGA)
****** C to intermediate language translation 5.3.1 (P_FPGA) took 0.2real, 0.2u + 0s
****** C to intermediate language translation 5.3.2 (P_GCPTS)
******* C to intermediate language translation 5.3.2.1 (P_GAA3)
******** C to intermediate language translation 5.3.2.1.1 (Loading)
******** C to intermediate language translation 5.3.2.1.1 (Loading) took 0real, 0u + 0s
[50 -> 89]
****** C to intermediate language translation 5.3.2 (P_GCPTS) took 1.1real, 1.1u + 0s
****** C to intermediate language translation 5.3.3 (P_MNPV)
****** C to intermediate language translation 5.3.3 (P_MNPV) took 0.7real, 0.7u + 0s
******** C to intermediate language translation 5.3.2.1.2 (P_GAA_SC)
********* C to intermediate language translation 5.3.2.1.2.1 (P_GAA_VI)
********* C to intermediate language translation 5.3.2.1.2.1 (P_GAA_VI) took 0real, 0u + 0s
********* C to intermediate language translation 5.3.2.1.2.2 (P_GAA_SDC)
********* C to intermediate language translation 5.3.2.1.2.2 (P_GAA_SDC) took 0real, 0u +
0s
********* C to intermediate language translation 5.3.2.1.2.3 (P_GAA_RS)
********* C to intermediate language translation 5.3.2.1.2.3 (P_GAA_RS) took 0real, 0u + 0s
******** C to intermediate language translation 5.3.2.1.2 (P_GAA_SC) took 0real, 0u + 0s
******* C to intermediate language translation 5.3.2.1 (P_GAA3) took 2.5real, 2.5u + 0s
***** C to intermediate language translation 5.3 (P_GA) took 3.7real, 3.7u + 0s
***** C to intermediate language translation 5.4 (P_AA)
****** C to intermediate language translation 5.4.1 (P_AC)
Some stats on points to analysis:
   Number of optimized point_to edges: 15
****** C to intermediate language translation 5.4.1 (P_AC) took 0real, 0u + 0s
***** C to intermediate language translation 5.4 (P_AA) took 0real, 0u + 0s
***** C to intermediate language translation 5.5 (P_PFF)
Found 12 polymorphic functions
***** C to intermediate language translation 5.5 (P_PFF) took 0real, 0u + 0s
***** C to intermediate language translation 5.6 (P_LGR)
***** C to intermediate language translation 5.6 (P_LGR) took 0real, 0u + 0s
**** C to intermediate language translation 5 (P_SFNPV) took 5real, 5u + 0s
**** C to intermediate language translation 6 (P_SP)
**** C to intermediate language translation 6 (P_SP) took 0.8real, 0.8u + 0s
**** C to intermediate language translation 7 (P_RB)
**** C to intermediate language translation 7 (P_RB) took 0real, 0u + 0s
**** C to intermediate language translation 8 (P_PA)
***** C to intermediate language translation 8.1 (P_ATA)
***** C to intermediate language translation 8.1 (P_ATA) took 0.1real, 0.1u + 0s
***** C to intermediate language translation 8.2 (P_AP)
***** C to intermediate language translation 8.2 (P_AP) took 0real, 0u + 0s
***** C to intermediate language translation 8.3 (P_ITFP)
***** C to intermediate language translation 8.3 (P_ITFP) took 0real, 0u + 0s
***** C to intermediate language translation 8.4 (P_CA)
****** C to intermediate language translation 8.4.1 (P_STS)
****** C to intermediate language translation 8.4.1 (P_STS) took 0.1real, 0.1u + 0s
****** C to intermediate language translation 8.4.2 (P_RR)
****** C to intermediate language translation 8.4.2 (P_RR) took 0real, 0u + 0s
Some stats on aliases computation:
   Number of aliases sets:          39
   Number of couples of aliases: 267
   Number of elements in the biggest alias sets: 1st=8, 2nd=7, 3rd=7, 4th=7, 5th=6
***** C to intermediate language translation 8.4 (P_CA) took 0.1real, 0.1u + 0s
**** C to intermediate language translation 8 (P_PA) took 0.3real, 0.3u + 0s
```

```
**** C to intermediate language translation 9 (P_SSet)
**** C to intermediate language translation 9 (P_SSet) took 0.6real, 0.6u + 0s
**** C to intermediate language translation 10 (P_O)
**** C to intermediate language translation 10 (P_O) took 0.5real, 0.5u + 0s
**** C to intermediate language translation 11 (P_G)
**** C to intermediate language translation 11 (P_G) took 0.3real, 0.3u + 0s
**** C to intermediate language translation 12 (P_TT)
**** C to intermediate language translation 12 (P_TT) took 0.1real, 0.1u + 0s
**** C to intermediate language translation 13 (P_VT)
**** C to intermediate language translation 13 (P_VT) took 0real, 0u + 0s
**** C to intermediate language translation 14 (P_PT)
***** C to intermediate language translation 14.1 (P_SPP)
****** C to intermediate language translation 14.1.1 (P_CSSIP)
****** C to intermediate language translation 14.1.1 (P_CSSIP) took 0real, 0u + 0s
***** C to intermediate language translation 14.1 (P_SPP) took 0real, 0u + 0s
***** C to intermediate language translation 14.2 (P_TP)
  - translating procedure assert (1 / 22)
  - translating procedure memset (2 / 22)
  - translating procedure floor (3 / 22)
  - translating procedure Fu_DriverInformationSystem_Init (4 / 22)
  - translating procedure FunctionalModel0209_initialize (5 / 22)
  - translating procedure FunctionalModel0209_terminate (6 / 22)
  - translating procedure printf (7 / 22)
  - translating procedure fflush (8 / 22)
  - translating procedure _init_globals_0 (9 / 22)
  - translating procedure _init_globals_0_1 (10 / 22)
  - translating procedure _init_globals_0_2 (11 / 22)
  - translating procedure main (12 / 22)
* warning, file: "ert_main.c", 72:37 :
    precision loss in read of FunctionalModel0209_M->errorStatus because
FunctionalModel0209_M may point to volatile data
* warning, file: "ert_main.c", 72:37 :
    precision loss in read of FunctionalModel0209_M->errorStatus because
FunctionalModel0209_M may point to volatile data
  - translating procedure __PST__MAIN__ENTRY__POINT__ (13 / 22)
Some stats on aliases use:
    Number of alias writes:          28
    Number of must-alias writes: 26
    Number of pma writes:            26
    Number of alias reads:           0
    Number of invisibles:            0
Stats about alias writes:
    biggest sets of alias writes: FunctionalModel0209_initialize:pVoidBlockIORegion (21),
memset:p_1 (5), FunctionalModel0209_M (1)
    procedures that write the biggest sets of aliases: FunctionalModel0209_initialize (22),
memset (5), fflush (1)
***** C to intermediate language translation 14.2 (P_TP) took 2.6real, 2.6u + 0s
**** C to intermediate language translation 14 (P_PT) took 2.7real, 2.7u + 0s
**** C to intermediate language translation 15 (P_IL)
***** C to intermediate language translation 15.1 (P_DRP)
***** C to intermediate language translation 15.1 (P_DRP) took 0real, 0u + 0s
***** C to intermediate language translation 15.2 (P_DR)
***** C to intermediate language translation 15.2 (P_DR) took 0real, 0u + 0s
***** C to intermediate language translation 15.3 (P_IGA)
***** C to intermediate language translation 15.3 (P_IGA) took 0.3real, 0.3u + 0s
***** C to intermediate language translation 15.4 (P_AG)
0 constructions broken due to gotos
```

```
***** C to intermediate language translation 15.4 (P_AG) took 0.2real, 0.2u + 0s
***** C to intermediate language translation 15.5 (P_CG)
***** C to intermediate language translation 15.5 (P_CG) took 0.2real, 0.2u + 0s
***** C to intermediate language translation 15.6 (P_R)
***** C to intermediate language translation 15.6 (P_R) took 0.2real, 0.2u + 0s
***** C to intermediate language translation 15.7 (P_PP)
* 0 pp, 7 ppp.
***** C to intermediate language translation 15.7 (P_PP) took 0.2real, 0.2u + 0s
***** C to intermediate language translation 15.8 (P_ICSP)
* 220 cd, 4048 cf, 0 rc, 0 ff, 0 ed, 0 cd.
***** C to intermediate language translation 15.8 (P_ICSP) took 1.6real, 1.6u + 0s
***** C to intermediate language translation 15.9 (P_ILA)
***** C to intermediate language translation 15.9 (P_ILA) took 0.2real, 0.2u + 0s
***** C to intermediate language translation 15.10 (P_PGC)
* 1665 tdl.
***** C to intermediate language translation 15.10 (P_PGC) took 0.6real, 0.6u + 0s
***** C to intermediate language translation 15.11 (P_ILA)
***** C to intermediate language translation 15.11 (P_ILA) took 0.1real, 0.1u + 0s
***** C to intermediate language translation 15.12 (P_PGC)
* 864 tdl.
***** C to intermediate language translation 15.12 (P_PGC) took 0.4real, 0.4u + 0s
***** C to intermediate language translation 15.13 (P_SULV)
***** C to intermediate language translation 15.13 (P_SULV) took 0.1real, 0.1u + 0s
***** C to intermediate language translation 15.14 (P_ICPP)
***** C to intermediate language translation 15.14 (P_ICPP) took 0.5real, 0.5u + 0s
***** C to intermediate language translation 15.15 (P_PP)
* 0 pp, 0 ppp.
***** C to intermediate language translation 15.15 (P_PP) took 0.1real, 0.1u + 0s
***** C to intermediate language translation 15.16 (P_SRC)
* 9 rcd, 0 tpd.
***** C to intermediate language translation 15.16 (P_SRC) took 0.4real, 0.4u + 0s
***** C to intermediate language translation 15.17 (P_SULV)
***** C to intermediate language translation 15.17 (P_SULV) took 0.1real, 0.1u + 0s
***** C to intermediate language translation 15.18 (P_SENUP)
12 empty procedure(s) removed
***** C to intermediate language translation 15.18 (P_SENUP) took 0real, 0u + 0s
***** C to intermediate language translation 15.19 (P_R)
***** C to intermediate language translation 15.19 (P_R) took 0.1real, 0.1u + 0s
**** C to intermediate language translation 15 (P_IL) took 8.5real, 8.5u + 0s
1 empty package(s) removed
**** C to intermediate language translation 16 (P_IPF)
92% init procedures removed
**** C to intermediate language translation 16 (P_IPF) took 0.1real, 0.1u + 0s
66% types removed
* assigns: 79% reduction
* asserts: 51% reduction
* total    : 87% reduction
************************************************************
***
*** C to intermediate language translation done
***
************************************************************
Ending at: Feb 9, 2009 14:58:8
User time for iabc-c2if: 35.8real, 35.8u + 0s
Starting at: Feb 9, 2009 14:58:8
************************************************************
***
```

```
*** Beginning Quick Software Safety Integration Analysis
***
**********************************************************
**** Quick Software Safety Integration Analysis 1 (MF)
**** Quick Software Safety Integration Analysis 1 (MF) took 0real, 0u + 0s
**** Quick Software Safety Integration Analysis 2 (interprocedural propagation)
**** Quick Software Safety Integration Analysis 2 (interprocedural propagation) took 0.2real,
0.2u + 0s

Generating GUI files
Checks statistics: (including internal files)
  - OVFL     => Green :        2, Orange :        0, Red :        0, Gray :        4     (100%)
  - NIP      => Green :        4, Orange :        0, Red :        0, Gray :        2     (100%)
  - NIVL     => Green :        3, Orange :        0, Red :        0, Gray :        7     (100%)
  - UNFL     => Green :        2, Orange :        0, Red :        0, Gray :        4     (100%)
  - OBAI     => Green :        0, Orange :        0, Red :        0, Gray :        1     (100%)
  - ZDV      => Green :        0, Orange :        0, Red :        0, Gray :        1     (100%)
  - IDP      => Green :        2, Orange :        1, Red :        0, Gray :        1     (75%)

TOTAL:       => Green :       13, Orange :        1, Red :        0, Gray :       20     (97%)

Number of NTL : 0
Number of NTC : 0
Number of UNR : 0


GUI files generation complete.

**********************************************************
***
*** Quick Software Safety Integration Analysis done
***
**********************************************************
Ending at: Feb 9, 2009 14:58:13
User time for quick: 5.4real, 5.4u + 0s
User time for polyspace-c: 112.6real, 112.6u + 0s


***
*** End of PolySpace Verifier analysis
***
```