ALTERNATIVE VEHICLE ELECTRONIC ARCHITECTURE FOR INDIVIDUAL WHEEL CONTROL

By

SETHA PAN-NGUM

BEng, MSc

A thesis submitted for the degree of Doctor of Philosophy in Engineering

University of Warwick

School of Engineering

August 2001

ABSTRACT

Electronic control systems have become an integral part of the modern vehicle and their installation rate is still on a sharp rise. Their application areas range from powertrain, chassis and body control to entertainment. Each system is conventionally control led by a centralised controller with hard-wired links to sensors and actuators. As systems have become more complex, a rise in the number of system components and amount of wiring harness has followed. This leads to serious problems on safety, reliability and space limitation. Different networking and vehicle electronic architectures have been developed by others to ease these problems. The thesis proposes an alternative architecture namely Distributed Wheel Architecture, for its potential benefits in terms of vehicle dynamics, safety and ease of functional addition. The architecture would have a networked controller on each wheel to perform its dynamic control including braking, suspension and steering.

The project involves conducting a preliminary study and comparing the proposed architecture with four alternative existing or high potential architectures. The areas of study are functionality, complexity, and reliability.

Existing ABS, active suspension and four wheel steering systems are evaluated in this work by simulation of their operations using road test data. They are used as exemplary systems, for modelling of the new electronic architecture together with the four alternatives. A prediction technique is developed, based on the derivation of software pseudo code from system specifications, to estimate the microcontroller specifications of all the system ECUs. The estimate indicates the feasibility of implementing the architectures using current microcontrollers. Message transfer on the Controller Area Network (CAN) of each architecture is simulated to find its associated delays, and hence the feasibility of installing CAN in the architectures. Architecture component costs are estimated from the costs of wires, ECUs, sensors and actuators. The number of wires is obtained from the wiring models derived from exemplary system data. ECU peripheral component counts are estimated from their statistical plot against the number of ECU pins of collected ECUs. Architecture component reliability is estimated based on two established reliability handbooks.

The results suggest that all of the five architectures could be implemented using present microcontrollers. In addition, critical data transfer via CAN is made within time limits under current levels of message load, indicating the possibility of installing CAN in these architectures. The proposed architecture is expected to be costlier in terms of components than the rest of the architectures, while it is among the leaders for wiring weight saving. However, it is expected to suffer from a relatively higher probability of system component failure.

The proposed architecture is found not economically viable at present, but shows potential in reducing vehicle wire and weight problems.

ACKNOWLEDGEMENTS

I am deeply grateful to my supervisor, R J Ball, for his elaborate guidance, patience and a lot of time, throughout the course of this project. The completion of this thesis would not have been possible without his support.

I am also grateful to the following people for their significant assistance to the project: Mr. David Carter for kindly explaining the air suspension control operation to me; University of Warwick inter-library loan staff for their efficient and friendly service to my numerous requests; The Royal Thai Government for their financial support.

I would also like to thank my parents for all my education and upbringing, Ya and Mai for their mental and nutritional supports, and Pan for her patience and encouragement.

DECLARATION

I declare that all the work described in this thesis was undertaken by myself unless otherwise acknowledged in the text, and that none of the work has previously been submitted for any academic degree. All sources of quoted information have been acknowledged by means of references.

Setha Pan-ngum

TABLE OF CONTENTS

CHAPTER 1 1

INTRODUCTION TO THE PROJECT 1

1.1 Background 1
1.2 Project Objectives 2
1.3 Possible Benefits of Distributed Wheel Controller Architecture 3
1.3.1 Vehicle Dynamics 3
1.3.2 Electronic Point of View 4
1.4 Methodology 5
1.5 Thesis Contents 6
1.6 References 7

CHAPTER 2 8

LITERATURE REVIEW 8

2.1 Suspension Control 8
2.1.1 Conventional Passive Suspension System 8
2.1.2 Semi-Active Suspension System 9
2.1.3 Active Suspension System 15
2.1.4 Suspension System Electronic Control Units 17
2.1.5 Current and Trend in Suspension System Research 17
2.2 Four Wheel Steering (4WS) 18
2.2.1 Types of 4WS Systems 19
2.2.2 4WS Control Algorithms 19
2.2.3 4WS Electronic Control Units 21
2.2.4 Status and Future Trend of 4WS 22
2.3 Electronics in Braking and Anti-lock Braking Systems (ABS) 23
2.3.1 Types of ABS 24
2.3.2 ABS Control Algorithms 24
2.3.3 ABS Electronic Control Units 28
2.3.4 Current Research on ABS 29
2.4 Microcontrollers 30
2.4.1 Introduction 30
2.4.2 Architectures 31
2.4.3 Microcontroller Trends in Automotive Electronics 37
2.5 Multiplexing and Controller Area Networks (CAN) 38
2.5.1 Multiplexing 38
2.5.2 CAN 40
2.5.3 CAN Microcontrollers 43
2.6 Automotive Electronic Architecture 45
2.6.1 Conventional Architecture 46
2.6.2 Centralised Controller 47
2.6.3 Peer to Peer Networking 48
2.6.4 Master-Slave Networking 49
2.7 References 51

CHAPTER 3 56

DERIVATION OF DIFFERENT AUTOMOTIVE ELECTRONIC CONTROL SYSTEMS & TEST DATA 56

3.1 EXEMPLARY SYSTEMS 56
3.1.1 Exemplary ABS System 57
3.1.2 Exemplary Suspension Control System 58
3.1.3 Exemplary 4WS System 59
3.2 EXEMPLARY DYNAMIC CONTROL SYSTEM FUNCTIONALITY 60
3.2.1 ABS Control Model 61
3.2.2 Active Suspension Control Model 63
3.2.3 4WS Control Model 65
3.3 ROAD TEST DATA 67
3.3.1 Data Preparation 68
3.3.2 Wheel Speed Data 70
3.3.3 Wheel Height Data 77
3.3.4 Vertical Acceleration Data 77
3.3.5 Longitudinal Acceleration Data 78
3.3.6 Lateral Acceleration Data 78
3.3.7 Steering Wheel Angle Data 80
3.3.8 Yaw Rate Data 82
3.3.9 Whole Vehicle Model 83
3.4 WHOLE VEHICLE SIMULATION 85
3.4.1 ABS Simulation 85
3.4.2 Active Suspension Simulation 88
3.4.3 4WS Simulation 90
3.5 SUMMARY 91
3.6 REFERENCES 91

CHAPTER 4 92

VEHICLE ELECTRONIC ARCHITECTURES UNDER CONSIDERATION 92

4.1 CONVENTIONAL CENTRALISED ARCHITECTURE (*ARCHITECTURE 1*) 92
4.2 CONVENTIONAL CENTRALISED WITH LIMITED CAN INTERACTION ARCHITECTURE (*ARCHITECTURE 2*) 93
4.3 TOTAL CENTRALISED ARCHITECTURE (*ARCHITECTURE 3*) 94
4.4 CONVENTIONAL CENTRALISED WITH FUNCTIONAL INTEGRATION ARCHITECTURE (*ARCHITECTURE 4*) 94
4.5 DISTRIBUTED WHEEL CONTROLLER ARCHITECTURE (*ARCHITECTURE 5*) 95
4.6 REFERENCES 96

CHAPTER 5 97

ELECTRONIC ARCHITECTURE FEASIBILITY STUDIES ON HARDWARE REQUIREMENTS AND CAN MESSAGE DELAY 97

5.1 ECU HARDWARE AND PERFORMANCE REQUIREMENTS 97
5.1.1 The Vehicle System Design Process 97
5.1.2 Hardware Performance Estimation of Known ECUs 98
5.1.3 Cruise control ECU Experiment 99
5.1.4 Air Suspension ECU Experiment 116
5.1.5 Difference Between Predicted and Actual Response Times 126
5.1.6 Correction (Multiplying) Factor 128
5.1.7 Conclusions from the Two Experiments 129

5.2 ECU PERFORMANCE REQUIREMENT OF DIFFERENT ARCHITECTURES 130
5.2.1 Microcontroller Selection Criteria 130
5.2.2 Alternative Architecture ECU Response Time Modelling 130
5.2.3 Active Suspension, 4WS, and ABS Memory and I/O Predictions 136
5.3 CONTROLLER AREA NETWORK (CAN) SIMULATION 140
5.3.1 Timing Aspects of CAN 140
5.3.2 Simulation Data 142
5.3.3 Simulation Package and Methods 144
5.3.4 Simulation Models 146
5.3.5 Simul8 Models 148
5.3.6 Simulation Run 150
5.3.7 Results and Analysis 151
5.4 REFERENCES 153

CHAPTER 6 155

ELECTRONIC ARCHITECTURE EFFECTS ON COMPLEXITY AND COSTS 155

6.1 EFFECT OF THE VEHICLE WIRING HARNESS ON DIFFERENT ARCHITECTURES 155
6.1.1 Wiring Models of Dynamic and Powertrain Control Systems 156
6.1.2 The Number of Wires and Their Estimated Total Length and Weight of Each Architecture 165
6.2 COST COMPARISON BETWEEN DIFFERENT ARCHITECTURES WIRING HARNESS 172
6.2.1 Component Cost Estimation 172
6.2.2 Analysis 177
6.3 FUTURE TRENDS AND OTHER EFFECTS ON SYSTEM COSTS 178
6.3.1 Future Trend Effects 178
6.3.2 ECU Development Cost 179
6.3.3 Assembly and Diagnostic Costs 180
6.4 SUMMARY 181
6.5 REFERENCES 182

CHAPTER 7 184

COMPONENT RELIABILITY PREDICTIONS OF ALTERNATIVE ELECTRONIC ARCHITECTURES 184

7.1 TYPES OF RELIABILITY PREDICTION TECHNIQUES 184
7.1.1 Similar Equipment and Similar Complexity Techniques 185
7.1.2 Prediction by Function Techniques 185
7.1.3 Part Count Techniques 186
7.1.4 Stress Analysis Techniques 186
7.2 RELIABILITY PREDICTION OF AUTOMOTIVE ELECTRONICS 187
7.3 RELIABILITY MODELLING OF DIFFERENT VEHICLE ELECTRONIC ARCHITECTURES 188
7.3.1 Reliability Modelling Based on AT&T Technique 189
7.3.2 Reliability Modelling Based on Automotive Reliability Data 194
7.3.3 Analysis 195
7.4 REFERENCES 196

CHAPTER 8 197

CONCLUSIONS 197

8.1 SIMULATION OF CONTROL SYSTEMS 198
8.2 MICROCONTROLLER SPECIFICATION 199
8.3 SIMULATION OF CAN MESSAGE TRANSFER 200
8.4 SYSTEM WIRING COMPLEXITY AND COMPONENT COSTS 200
8.5 ELECTRONIC COMPONENT RELIABILITY 201
8.6 DISTRIBUTED WHEEL CONTROLLER ARCHITECTURE 202
8.7 FURTHER WORK 203
8.8 REFERENCES 206

APPENDIX A PREDICTION OF ECU CONTROL PSEUDO CODE AND ASSEMBLY PROGRAMS 207

CRUISE CONTROL 207
Throttle Demand Calculation Pseudo Code 207
Throttle Demand Calculation Predicted Assembly Code 207
Predicted Assembly Programs of Other Processes 212
Response Time Prediction 213
AIR SUSPENSION 214
Air Suspension Flow Charts 214
Air Suspension Predicted Assembly Code 222
ABS 234
ABS Flow Chart 234
ABS Pseudo Code 235
4WS 237
4WS Flow Chart 237
4WS Pseudo Code 238
ACTIVE SUSPENSION 240
Active Suspension Flow Chart 240
Active Suspension Pseudo Code 241
PREDICTED RESPONSE TIME OF ALTERNATIVE ARCHITECTURE ECUs 243
Conventional Centralised, Conventional Centralised with Limited CAN Interaction, Conventional Centralised with Functional Integration Architectures (Architecture 1, 2, 4) 243
*Total Centralised Architecture (*Architecture 3*) 247*
Distributed Wheel Architecture (Architecture 5) 248

ROM ESTIMATION 251

OPERAND COUNT 251
Active Suspension 251
4WS 252
ABS 252
Distributed Wheel Architecture 253

APPENDIX B VEHICLE SIGNAL MODELS FOR CAN SIMULATION 255

APPENDIX C IN-VEHICLE SIGNAL LIST FOR VEHICLE WIRING MODELS 259

APPENDIX D SYSTEM COMPONENT COST CALCULATION 262

APPENDIX E ALTERNATIVE ARCHITECTURE RELIAIBILITY MODELLING 263

REFERENCES 265

LIST OF FIGURES

Figure 1.1 Outline of Distributed Wheel Controller Architecture 2

Figure 1.2 Vehicle g-g diagram 4

Figure 2.1 Adaptive damping suspension system layout 10

Figure 2.2 Adaptive damping suspension block diagram 10

Figure 2.3 Adaptive damping suspension functional block diagram 11

Figure 2.4 Self levelling suspension system layout 12

Figure 2.5 Self levelling suspension block diagram 12

Figure 2.6 Series active suspension block diagram 13

Figure 2.7 Parallel Active Suspension 14

Figure 2.8 Anti-roll System Control Block Diagram 14

Figure 2.9 Active suspension system layout 16

Figure 2.10 Active suspension system input and output signals 16

Figure 2.11 Acceleration control block diagram of active suspension 17

Figure 2.12 A typical front and rear wheel steering characteristic of a 4WS 18

Figure 2.13 Rear wheel steering characteristics of a simple 4WS system 20

Figure 2.14 Proportional control block diagram of a 4WS system 20

Figure 2.15 4WS with yaw rate feedback control system diagram 21

Figure 2.16 A 4WS ECU structure 22

Figure 2.17 Relationship between slip and tyre-road coefficient of friction 25

Figure 2.18 Simplified model of ABS electronic control applied to some three and four-channel systems 26

Figure 2.19 Typical ABS control cycles on high tyre-road friction coefficient road 27

Figure 2.20 Simplified model of an ABS ECU 28

Figure 2.21 Microcontroller simple block diagram 31

Figure 2.22 Microcontroller implementation growth in modern vehicles 38

Figure 2.23 System block diagram of high-speed CAN network 40

Figure 2.24 CAN DATA FRAME message format 42

Figure 2.25 Conventional centralised architecture 46

Figure 2.26 Centralised controller architecture 47

Figure 2.27 Peer to Peer network architecture 48

Figure 2.28 Master Slave network architecture 50

Figure 3.1 Simplified electrical diagram of an exemplary ABS system 57

Figure 3.2 Simplified electrical diagram of an exemplary suspension control system 59

Figure 3.3 Simplified electrical diagram of an exemplary 4WS control system 60

Figure 3.4 ABS control cycle 62

Figure 3.5 *Simulink* model of the ABS system 62

Figure 3.6 Exemplary active suspension control block diagram 64

Figure 3.7 *Simulink* model of active suspension system 64
Figure 3.8 Exemplary 4WS system control block diagram 66
Figure 3.9 *Simulink* model of 4WS system 67
Figure 3.10 Map of Rover's Emissions test track at Gaydon 68
Figure 3.11 Vehicle speed data produced from road test 68
Figure 3.12 Vehicle speed and distance travelled with track position indication 70
Figure 3.13 Fishbone diagram of the causes of wheel speed difference 71
Figure 3.14 Friction force against wheel slip graph 73
Figure 3.15 Normalised longitudinal force against normalised wheel slip plot 74
Figure 3.16 Four wheel speed data 76
Figure 3.17 Wheel height and vertical acceleration data 77
Figure 3.18 Model used to generate vertical acceleration data from the wheel height data 78
Figure 3.19 Longitudinal acceleration data 78
Figure 3.20 Lateral acceleration data 79
Figure 3.21 Relationship between maximum front steering angle and vehicle speed 81
Figure 3.22 Front wheel steering angle data 82
Figure 3.23 Yaw rate data 83
Figure 3.24 *Simulink* vehicle model of the dynamic control systems 84
Figure 3.25 Four wheel speed sensor inputs 85
Figure 3.26 ABS responses 86
Figure 3.27 Modified front left wheel speed to show sign of wheel locking 87
Figure 3.28 ABS responses to front left wheel locking 87
Figure 3.29 Active suspension inputs 88
Figure 3.30 Active suspension responses 89
Figure 3.31 4WS inputs and responses 90
Figure 4.1 Conventional centralised architecture 93
Figure 4.2 Conventional centralised with limited CAN interaction architecture 93
Figure 4.3 Total centralised architecture 94
Figure 4.4 Conventional centralised with functional integration architecture 95
Figure 5.1 Cruise control state transition diagram 100
Figure 5.2 Cruise control motor and solenoid operations 101
Figure 5.3 Cruise control operation flowchart 103
Figure 5.4 Cruise control experiment circuit connection 105
Figure 5.5 Cruise control microcontroller connections 106
Figure 5.6 Waveforms when SET is pressed 107
Figure 5.7 Pulse response time to SET command 108
Figure 5.8 Solenoid response time 108
Figure 5.9 Motor response time 108
Figure 5.10 Scatter plot of pulse response time 109
Figure 5.11 Scatter plot of solenoid response time 110

Figure 5.12 Signal waveforms when 'RES' is pressed 110
Figure 5.13 Response time to SET (after cruise disengaged) 111
Figure 5.14 Response time to manually controlled 'RES' (disengage) 112
Figure 5.15 Response time to 'RES' (disengage), controlled by solenoid signal 113
Figure 5.16 Air suspension software structure 120
Figure 5.17 Air suspension measurement circuit connections 122
Figure 5.18 Air suspension microcontroller connections 123
Figure 5.19 Response to UP demand 124
Figure 5.20 Response to footbrake during height state change 124
Figure 5.21 Response to DOWN 125
Figure 5.22 Percent of RTOS overhead on CPU 133
Figure 5.23 A series of periodic signal getting transmitted 141
Figure 5.24 Number of signals of each type 143
Figure 5.25 Percentage of bandwidth of different types of signals per total bandwidth 144
Figure 5.26 CAN simulation model from literature 145
Figure 5.27 *Simul8* model of *Architecture 4* 149
Figure 5.28 Timing diagram of a 10 ms period signal generation 150
Figure 5.29 Frequency Distribution of CAN Transmission Time of 5 ms Period Class C Signals 151
Figure 5.30 Frequency Distribution of CAN Transmission Time of 10 ms Period Class C Signals 152
Figure 6.1 Growth in the number of powertrain ECU I/Os against calendar year 156
Figure 6.2 Wiring model of *Architecture 1* 161
Figure 6.3 Wiring model of *Architectures 2,4* 162
Figure 6.4 Wiring model of *Architecture 3* 163
Figure 6.5 Wiring model of *Architecture 5* 164
Figure 6.6 Vehicle zonal diagram 166
Figure 6.7 ECU wiring from passenger compartment to door panels 167
Figure 6.8 ECU wiring from passenger compartment to engine bay area 167
Figure 6.9 Plot of number of ECU components and I/O pins 175
Figure 6.10 System costs of different architectures 177
Figure 7.1 A bathtub curve representing typical component failure rate 188
Figure 7.2 Hazard rate model developed by AT&T [10] 190
Figure 7.3 Projected number of peripheral components per ECU pins 192

LIST OF TABLES

Table 5.1 Response time prediction results of *Architectures 1, 2 and 4* 134

Table 5.2 Response time prediction results of *Architecture 3* 135

Table 5.3 Response time prediction results of *Architecture 5* 135

Table 5.4 ROM, RAM and I/O requirement Predictions of all the architectures 139

Table 5.5 Signal Bandwidth of all the Architectures 147

Table 5.6 Worst Case Delay of Class C Signals of all the Architectures 153

Table 6.1 List of sensors and actuators in the models 158

Table 6.2 Total number of wires in different vehicle electronic architectures 165

Table 6.3 Wiring length classification of different vehicle electronic architectures 166

Table 6.4 Length of four wire types and total lengths of wires in different vehicle electronic architectures 169

Table 6.5 CAN wiring lengths of the four vehicle electronic architectures 170

Table 6.6 Total number, lengths and weights of wires in different vehicle electronic architectures .. 171

Table 6.7 Wiring harness costs of different electronic architectures 173

Table 6.8 Estimated microcontroller costs 174

Table 6.9 Microcontrollers and peripheral ECU component costs 176

Table 6.10 Combined sensor and actuator costs 176

Table 6.11 Overall system costs 177

Table 7.1 Reliability modelling of alternative architectures using AT&T data 193

Table 7.2 Reliability modelling of alternative architectures using automotive reliability data 194

ABBREVIATIONS

4WS	Four Wheel Steering System
A/D	Analogue-to Digital
ABS	Anti-lock Braking System
ADC	Analogue-to-Digital Converter
ALU	Arithmetic Logic Unit
ASE	Air Suspension/EVO Steering
ASR	Anti-Spin Regulation (Traction Control System)
AVCS	Advanced Vehicle Control System
BECM	Body Electrical Control Module
BT	British Telecom
CAN	Controller Area Network
CCM	Climate Control Module
CCS	Cruise Control System
CISC	Complex Instruction Set Computer
CPU	Central Processing Unit
D/A	Digital-to-Analogue
EAROM	Electrically Alterable Read Only Memory
ECU	Electronic Control Unit
EEPROM	Electrically Erasable and Programmable Read Only Memory
EMC	ElectroMagnetic Compatibility
EMI	ElectroMagnetic Interference
EMS	Engine Management System
FIT	Failure In Time
FMEA	Failure Mode and Effect Analysis
HSIO	High Speed Input/Output
I/O	Input/Output
IC	Integrated Circuit

ICD	Instrument Cluster Display
ICM	Ignition Control Module
ISO	International Standards Organisation
LED	Light Emitting Diode
LSIO	Low Speed Input/Output
MTBF	Mean Time Between Failure
MTTF	Mean Time To Failure
OEM	Original Equipment Manufacturer
OSEK	Open Systems and Corresponding Interfaces for Automotive Electronics
OSI	Open Systems Interconnect
OTP	One-Time Programmable
PC	Personal Computer
PCM	Powertrain Control Module
PID	Proportional, Integral, Differential
PSS	Passenger Safety System
PWM	Pulse Width Modulation
RAM	Random Access Memory
RISC	Reduced Instruction Set Computer
ROM	Read Only Memory
RTOS	Real-Time Operating System
RWM	Read/Write Memory
SAE	Society of Automotive Engineers
TTP	Time Triggered Protocol
VAN	Vehicle Area Network
W	Width
WB	Wheelbase

CHAPTER 1

INTRODUCTION TO THE PROJECT

1.1 Background

Electronic systems are becoming very significant parts of the modern road vehicle. Their application areas range from powertrain, chassis and body control through to heating, ventilation and entertainment. Their functions are expanding into new areas to include navigation, crash avoidance, etc. [1-4]. The existing systems are also being constantly improved to meet customer expectations and future regulations. Consequently, both the complexity and cost of electronic systems per vehicle is rising and expected to rise further [3].

This increase in complexity in turn gives rise to increased numbers of sensors and actuators and also connections, and the amount of wiring harness required. Modern high-end vehicles contain a few kilometres of wiring, weighing at least 22 kilograms [7,8]. The more complex the systems become, the more likely it is that data needs to be exchanged amongst them. This is evident now from examples such as gearbox-engine links that improve gearchange shift quality and steering-suspension system links that stiffen the suspension during high 'g' manoeuvres. The cumulative effect of this type of link is in larger quantities of data being transferred between the electronic control units (ECUs).

Dynamic control systems such as Anti-lock Braking System (ABS), Traction Control, Suspension Control, and 4-wheel drive and 4-wheel steering, all operate via the wheels at the four corners of a vehicle. Conventionally, a centralised controller is used for each function, with hard-wired links to sensors and actuators distributed around the vehicle.

The increasing complexity of electronics systems and their interconnections in modern vehicles is a serious concern [9]. It can lead to reliability and safety problems, since wiring is still a major source of faults in a vehicle. In order to counter the problem, networking architectures such as Controller Area Network (CAN) have been developed to simplify wiring harnesses and enable data links between systems. It is in addition to these data networks that a number of different electronic architectures, described in the following chapter, are under consideration to help ease the problem.

1.2 Project Objectives

The aim of this work is to evaluate alternative automotive architectures for dynamic control electronics. The work adopts a system-level approach, in acknowledgement of the different factors that influence the adoption of new vehicle systems. Specifically, the functionality, reliability and cost of the alternative architectures are evaluated to compare their practicality and commercial potential.

A specific arrangement proposed here is that of a distributed architecture, which is anticipated to reduce the problem of increasing wiring harness and achieve further benefits, especially in terms of vehicle handling and electronics. This arrangement follows the concept of independent wheel control systems for suspension [19] and brake [10,18,20]. The proposed architecture would contain an integrated controller at each wheel and a central controller. These 'Wheel Controllers' would take control of the dynamics of each wheel, concerning traction, braking and suspension controls, with data connections via a networking protocol to other controllers. The outline of the system is shown in Figure 1.1.

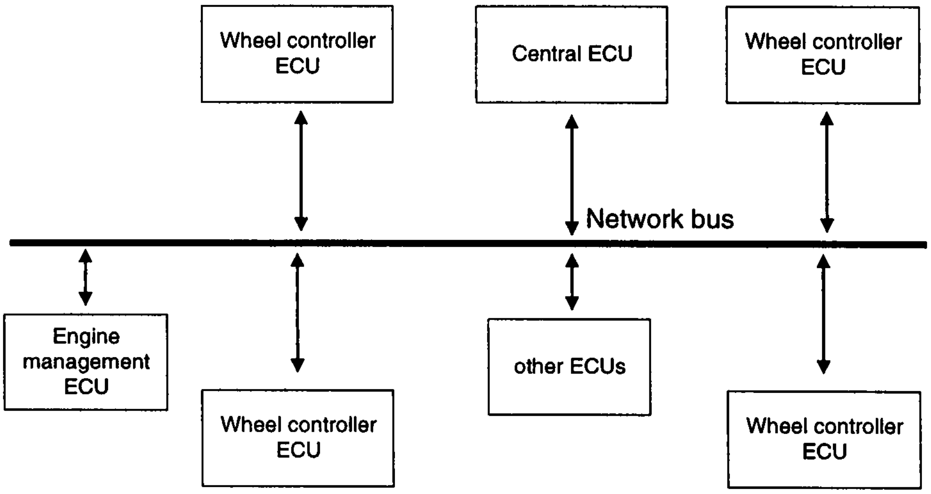


Figure 1.1 Outline of Distributed Wheel Controller Architecture

Specific objectives of this work are:

- to simulate the system operation using historical data from an instrumented vehicle

- to derive the functional specification of the dynamics control systems in a near-future vehicle and from this to estimate the processor specification required to support it
- to study the feasibility of installing a CAN network in the alternative architectures, in terms of message delay time
- to compare the desirability of the alternative architectures to the vehicle manufacturer in terms of cost and reliability
- to evaluate the viability of the distributed wheel control architecture on the above terms.

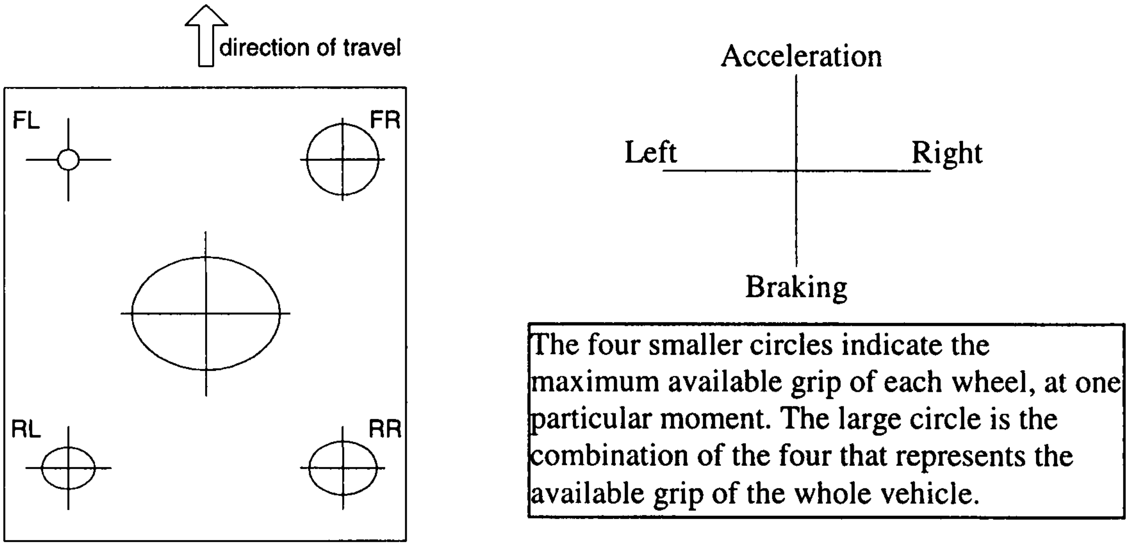
1.3 Possible Benefits of Distributed Wheel Controller Architecture

The foreseeable advantages of the proposed architectures will be in the following areas:

1.3.1 Vehicle Dynamics

In term of vehicle dynamics, a vehicle characteristic can be represented by the *g-g diagram* [5] shown in Figure 1.2, which is the maximum grip per unit load that four tyres can produce in any direction. This available force depends on the tyre-road friction and the load on the wheels, and changes from moment to moment. High performance cars tend to have a large diagram boundary. Although the diagram suggests maximum attainable force, normal vehicles cannot reach this limit. A skilled driver is often able to sense any reduction in grip, but is unable to redistribute the vehicle forces between the four wheels. One of the reasons is due to suspension geometry which affects wheel orientations and hence tyre forces. Imperfect brake balance between front and rear could also prevent both tyres from reaching the friction boundary simultaneously [5]. The available grip of each wheel cannot be measured in real-time, due to its dependability on factors such as road surfaces and tyre conditions. However, a system is being proposed that would sense and improve vehicle handling performance and stability near the limits of tyre-road grip, by distributing forces at each wheel accordingly [11,17]. Other systems are also intended to combine operations of brake, suspension and steering controls on each wheel, to improve vehicle handling [12-16].

The electronic architecture proposed here would implement this combined operation in a controller-per-wheel system, each controller sensing the grip when nearing the limit and communicating with the others in order to distribute the vehicle suspension, traction and cornering forces optimally.



The four smaller circles indicate the maximum available grip of each wheel, at one particular moment. The large circle is the combination of the four that represents the available grip of the whole vehicle.

Figure 1.2 Vehicle g-g diagram

1.3.2 Electronic Point of View

1.3.2.1 Cost

The component cost of the system is made up of ECU, wiring, sensor and actuator costs. The potential cost saving of the proposed architecture will be as a result of reducing the size of the wiring harness, which is currently the second most expensive item in a vehicle, after the engine [6]. The wiring reduction is due to networking and close proximity between a wheel controller and its sensors and actuators.

Balanced against this will be the potential costs of the four Wheel Control ECUs. The processing power, memory requirements and interface specifications of these are derived by the author to allow a comparison to be made with other architectures.

1.3.2.2 Safety Features

In individual wheel brake-by-wire control systems that have been proposed to date [18], a stated potential benefit is that a specified set of failures of electronics or mechanical components at one wheel could be compensated for by actions of the other three Wheel Control ECUs, as all are linked.

Similarly, this fault tolerance could be applied to other faulty cases. For instance, a deflating tyre could be identified and the load on the tyre redistributed.

Elaborate diagnostic records of sensors and actuators at each corner could be kept by each processor. The information could be of use in the future for trouble-predicting functions, to give advanced warning to the driver before a component actually breaks down [1].

1.4 Methodology

The work will firstly establish the likely electronic content of the dynamic control systems of a typical near-future vehicle. The electronic systems considered will be the Anti-lock Braking System (ABS), Suspension Control, and Four-Wheel Steering (4WS), which are collectively responsible for the vehicle's dynamics. These systems are to be considered together in a total of five potential architectures. These are: *Conventional Centralised* (*Architecture 1*), *Conventional Centralised with Limited CAN* (*Architecture 2*), *Total Centralised* (*Architecture 3*), and *Conventional Centralised with Functional Integration* (*Architecture 4*), as described in the following chapter, along with the proposed *Distributed Wheel Architecture* (*Architecture 5*) will be considered as the likely architectures of the future. The same electronic systems will be arranged into a vehicle model according to each architecture in turn. These models will form the basis for a study of the feasibility and characteristics of the architectures described in the objectives above, allowing a comparison of the potential benefits of each.

1.5 Thesis Contents

Chapter 2 contains literature survey on the automotive electronic architectures, mentioned in the thesis, and also the current developments and trends in dynamical control systems involved, which are ABS, suspension control and 4WS. Also included in Chapter 2 are current and future states of microcontrollers, which perform all the system electronic control, and introduction to Controller Area Network (CAN), which is a widely installed network architecture.

Chapter 3 describes the derivation of the exemplar dynamic control systems, which are ABS, active suspension and 4WS systems. The simulation using road test data to examine their functionality is described.

Chapter 4 explains the structures of the five vehicle electronic *Architecture*s, which are studied and compared in this project.

Chapter 5 includes the technique developed to predict the Electronic Control Unit (ECU) performance, from software specifications. The comparison between the predictions and experimental measurements, and the resulting derivation of a correction factor are explained. The performance prediction of the ECUs of all five *Architectures* is described, followed by the specifications of the required microcontrollers for the control tasks. This completes the architecture feasibility study on control hardware. The other feasibility study on CAN message delay, associated with each architecture, and its simulation results are then described.

Chapter 6 contains the wiring complexity and component cost estimation of the alternative architectures, focusing upon the wiring harness and ECU electronic contents. The effects of future trends and other factors on costs are also discussed.

Chapter 7 compares the architectures' characteristics on reliability, from modelling.

Chapter 8 draws the conclusions on the work and gives suggestions for further work.

1.6 References

1. *Kobayashi K, et al.* **Diagnostics for vehicle electronics present and future** Proceedings to the 1992 International Congress on Transportation, 1992
2. *Furukawa Y* **The direction of the future automotive safety technology** Proceedings to the 1992 International Congress on Transportation, 1992
3. *Ohr S* **Safety and security spearhead changes in automotive electronics** Computer Design, January 1996
4. *Reichart G* **Driver assistance - premises and promises** IMechE (C498/1/184/95), 1996
5. *Milliken W, et al.* **Race car vehicle dynamics** SAE, 1995
6. *McLaughlin R, et al.* **A feasibility study of CAN technology in body electronic control systems** IMechE Autotech 95, 1995
7. *Fenton J* **Focus on networking of on-board vehicle electronic systems** Automotive Engineer, June/July 1996
8. *Schreffler R* **Wire Harnesses of the Future** Automotive Industry, Feb 1997
9. *Inoue Y, et al.* **Multiplex Systems for Automotive Integrated Control** SAE No.930002
10. *Advanced Vehicle Systems Division, Motorola Semiconductor Products Sector* **'By Wire' Technology** Motorola, 2000
11. *van Zanten A, et al.* **Simulation for the Development of the Bosch-VDC** SAE No.960486
12. *Wallentowitz H* **Scope for the Integration of Powertrain and Chassis Control Systems: Traction Control - All-Wheel Drive - Active Suspension** SAE No.901168
13. *Hurdwell R, et al.* **Active Suspension and Rear Wheel Steering Make Powerful Research and Development Tools** SAE No.930266
14. *Sato H, et al.* **Development of Four Wheel Steering System Using Yaw Rate Feedback Control** SAE No.911922
15. *Stevens S* **The Development and Testing of an Integrated Systems Demonstrator Vehicle** IMechE C498/35/157/95, 1996
16. *Fruechte R, et al.* **Integrated Vehicle Control** 39th IEEE Vehicular Technology Conference, 1989
17. *Van Zanten, et al.* **Vehicle Dynamic Control** Automotive Engineering International, February 1999
18. *Bannatyne R* **Electronic Braking Control Developments** Automotive Engineering International, February 1999
19. *Ito H, et al.* **Controller for Experimental Vehicle Using Multi-Processor System** SAE No. 910086
20. *Hedenetz B, et al.* **Brake-By-Wire Without Mechanical Backup by Using a TTP-Communication Network** SAE No.981109

CHAPTER 2

LITERATURE REVIEW

This chapter provides the background to the choice of the individual vehicle dynamic control systems that are considered in this thesis. Additionally, the technologies available within the ECU are reviewed, with the aim of identifying the likely 'state-of-the-art' for a near-future ECU.

2.1 Suspension Control

Suspension has a crucial role in ride comfort and handling of a vehicle. Its functions are to isolate the vehicle body and its components from any irregularities of the roads, and to keep the tyres in contact with the ground at all time [1]. These, however, must be done without degrading stability and steering of the vehicle [2]. Ideally, a vehicle body should be level all the time, while suspension and wheels move vertically according to road irregularities. There are also other functions of the suspension as follows [3]:

- counteracting longitudinal, lateral forces and also braking and driving torque by the tyres
- opposing vehicle body roll
- maintaining proper steer and camber attitudes of wheels to road surface

2.1.1 Conventional Passive Suspension System

This type of system includes components that only react to the forces acting on the suspension, and hence is called passive [4]. The main component is the spring, which acts to oppose the suspension movement when a vehicle encounters a rough road surface. The other major element is a damper. The damper's function is to smooth out the oscillatory movement of the body/spring system.

The design of conventional suspension has traditionally been a compromise between ride and handling [5]. For example, a car with a high suspension spring stiffness tends to have higher stability than the one with lower stiffness, but the ride is not as smooth.

With advances in technology, electronic-controlled suspensions have been developed to improve the compromise inherent in conventional systems.

2.1.2 Semi-Active Suspension System

This is a system that still contains passive elements (spring and damper) but has additional controllable components for dynamic improvement in various driving situations. The controllable or semi-active elements may be either an air spring or adaptive damper [4]. Semi-active suspensions require less power and are cheaper than fully-active ones, but generally give inferior performance [8].

The typical semi-active system consists of several sensors depending on types of systems and an ECU [6, 7]. The sensors include wheel height sensors, vehicle speed sensor and a steering angle sensor.

According to the definition of active suspension by Milliken [5], many so-called active suspension systems, which still contain passive components, will be classified in this thesis as semi-active. Currently, the following systems would be categorised as semi-active:

- adaptive damping
- self-levelling
- series active
- parallel active
- anti-roll control

Adaptive damping and self-levelling systems are the most commonly implemented semi-active systems in cars at present [18]. These will now be discussed in more detail.

2.1.2.1 Adaptive Damping

An example of a system for which published component details and operating functions are available, is Toyota Electronic Modulated Suspension shown in Figure 2.1 [7]. It was first developed in 1983 and had been through various changes and improvements up until the publication year of 1991. It sets the damping force to one of several fixed values, in order to optimise riding comfort and stability according to the selected ride mode. The block diagram of the system is displayed in Figure 2.2 [7].

Here, the ECU adjusts the damping hardness according to the measured wheel height together with braking and steering information. The anti-roll function operates when wheel height and steering angle changes indicate body roll, with the ECU increasing the damping force to resist the movement.

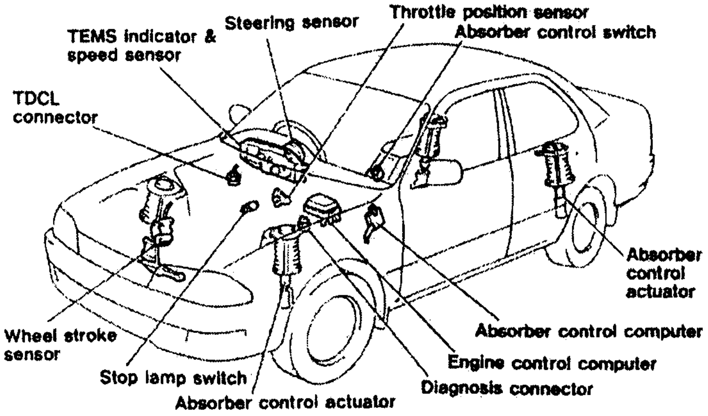


Figure 2.1 Adaptive damping suspension system layout

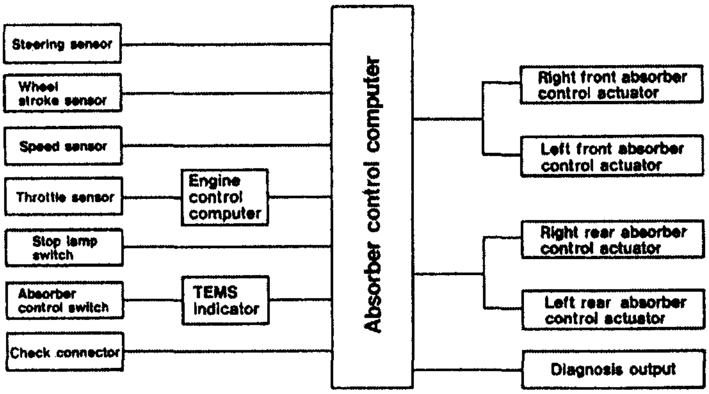

Steering sensor
Wheel stroke sensor
Speed sensor
Throttle sensor
Engine control computer
Stop lamp switch
Absorber control switch
TEMS indicator
Check connector
Absorber control computer
Right front absorber control actuator
Left front absorber control actuator
Right rear absorber control actuator
Left rear absorber control actuator
Diagnosis output

Figure 2.2 Adaptive damping suspension block diagram

The typical system functional block diagram including sensor information is shown in Figure 2.3 [7].

Figure 2.3 shows the control parameters, which are entirely a combination of vehicle sensor signals. The throttle data and the stop lamp data are used to adjust the setting of damping mode to minimise squat and dive, when accelerating and braking respectively. The steering sensor provides data for the anti-roll control function. In this particular system, feedback from the wheel stroke sensors is also incorporated with the above information to further enhance ride and handling. An example is the anti-dive function. A simple system would react to dive at high speed by increasing the damping. With the addition of wheel stroke data, this controller will also counteract dive at low speed, by detecting when the front wheel heights are low, indicating a dive.

The ECU processes these signals, and is capable of producing commands to the actuators at the interval at least equal to their operational response time of 10-20 ms.

Another system called a shock absorber control system, which was published in 1985, incorporates acceleration-deceleration sensor to aid vehicle movement data, and the ultrasonic road sensor to provide road condition data [6,13]. Another variation of the Toyota adaptive damping system omitted all the sensors, and only requires force reacted from road surface data, from the 4 piezoelectric sensors on shock absorbers for adaptive mode selection [6,14].

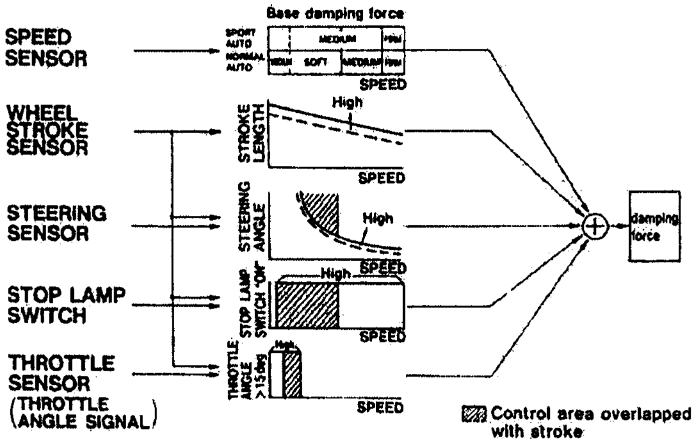


Figure 2.3 Adaptive damping suspension functional block diagram

2.1.2.2 Self Levelling System

This system functions to keep vehicle body height constant above the road surface, and also allows a low spring constant for ride comfort, regardless of load conditions [6]. The system is often combined with adaptive dampers [16]. Hence, on varying driving conditions and road surfaces, the combination of systems can adjust damping force and spring rate accordingly [6]. Self levelling is described here separately from adaptive damping because some systems perform only the self levelling task without damping control [15].

There are currently two implemented systems with published information, one developed by Rover in early 1990s [15] and the other described in [6] with no manufacturer information. Both of them use air springs as the self-levelling actuators. Examples of the system layout and block diagram are shown in Figure 2.4 and 2.5 [6].

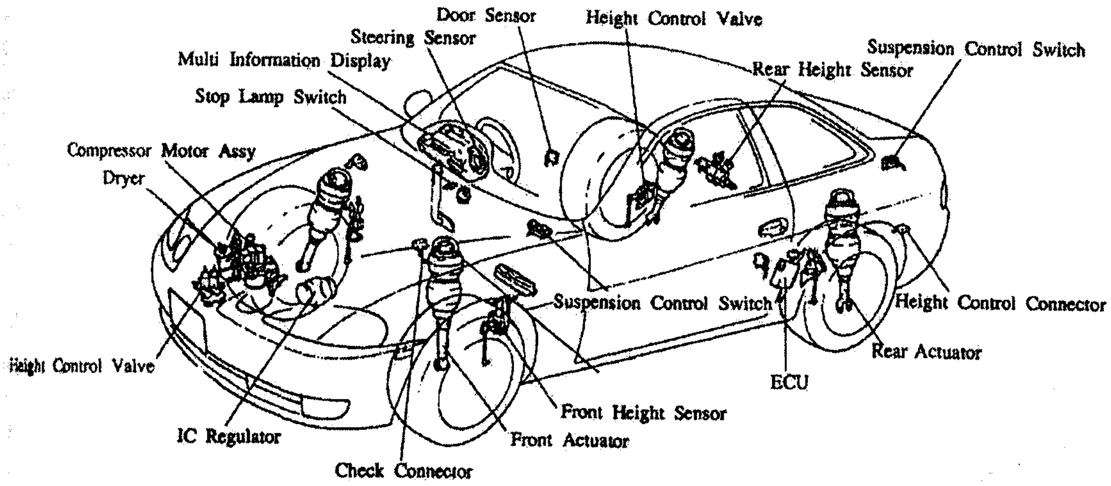


Figure 2.4 Self levelling suspension system layout

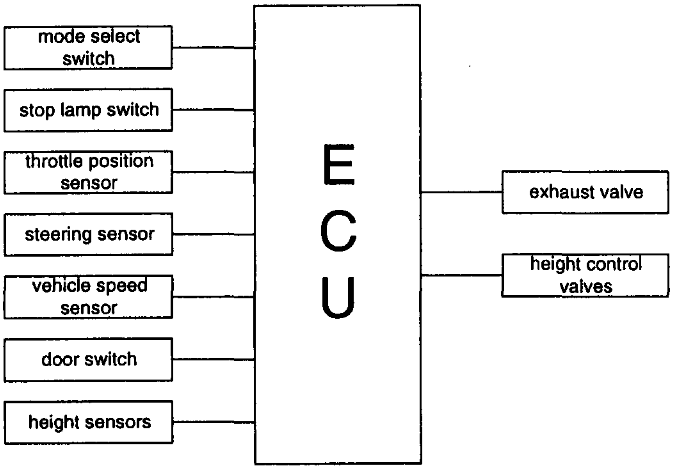


Figure 2.5 Self levelling suspension block diagram

This particular system has a combined functions of adaptive damping with the control of shock absorbers and air springs, and self levelling with 3 ride height level capability. As illustrated in Figure 2.5, this type of semi-active suspension requires similar types of sensor to the adaptive damping. The system in Figure 2.4 performs similar functions to a typical adaptive damping system with an added ability to raise and lower the vehicle body height to suit ride conditions. To improve ride and handling, these two functions are frequently combined into one system.

2.1.2.3 Series Active System

This semi-active suspension is sometimes called slow active suspension system [5,16]. As the name suggests, the system contains a conventional spring in series with a hydraulic actuator [17]. The actuator controls the ride at low frequencies and leaves the high frequency actions to the passive element, hence the name *slow active* [16].

A typical system control block diagram is as shown in Figure 2.6 [16].

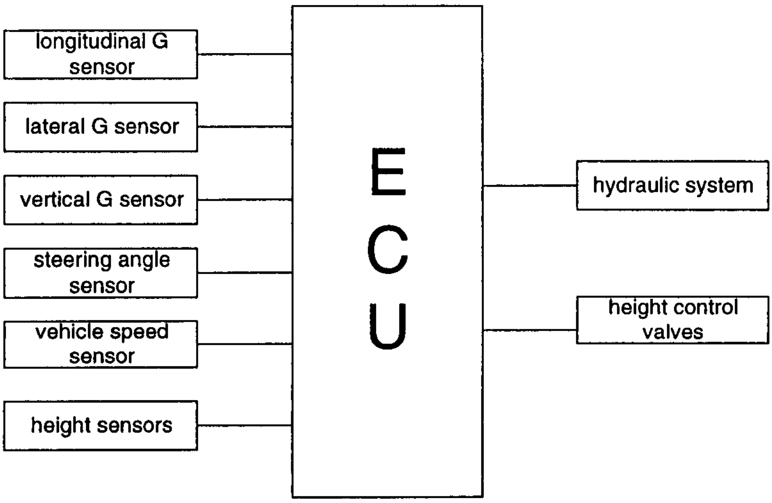


Figure 2.6 Series active suspension block diagram

The system contains a Proportional-Integral-Differential (PID) controller to correct the wheel height to the target wheel height [16]. PI feed-back vertical acceleration control is also applied to compensate for the lag of actuator response, for ride comfort. Finally, longitudinal and lateral acceleration sensors, together with vehicle speed and steering angle sensors, are used to supply data for the control of pitch and roll.

This system can attain better ride than adaptive damping with an extra benefit of being able to keep the vehicle level during cornering, braking and accelerating [18].

2.1.2.4 Parallel Active System

As the name suggests, the system is characterised by an actuator operating in parallel with a passive spring as shown in Figure 2.7 [18]

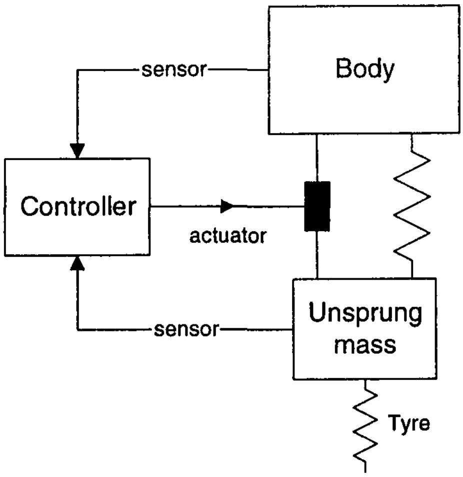


Figure 2.7 Parallel Active Suspension

The system is similar to active suspension but with the addition of passive springs to support the vehicle mass. The actuators, therefore, have less load to act upon and hence have a lower power consumption [17]. There is no information on system ECU, sensors and actuators, or control algorithm found in literature.

2.1.2.5 Anti-roll Control

Most modern cars have an anti-roll bar, usually a torsional spring, connecting the front wheels. Some cars also have one fitted on the rear wheels. The effect of the anti-roll bars is realised when the two front or rear wheels move vertically in opposite directions, which occurs during cornering, causing the body to roll. The bars will react against this roll. To reduce the vehicle roll, and hence improve ride, when cornering, an actuator is placed in series with the anti-roll bars. It can exert torque on the bars to counteract the roll [18]. The result is a better ride. The control block diagram is presumed to be as shown in Figure 2.8.

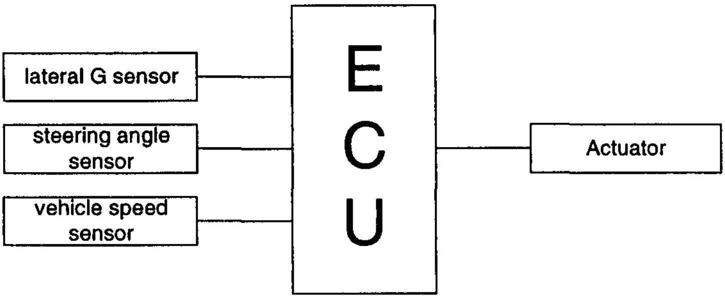


Figure 2.8 Anti-roll System Control Block Diagram

The anti-roll control system has no benefit in a straight line driving. The logical combination of anti-roll system and another semi-active suspension could improve ride and handling characteristics closer to those of active suspension with lower cost and power consumption [18].

2.1.3 Active Suspension System

An active system is defined here as a control system which has external power added to it. The fully active suspension system is totally dependent on the powered control elements [5]. It exerts forces via the actuators between vehicle body and wheels irrespective of current dynamics of the vehicle elements [4]. With present technology, the only practical active suspension system is electro-hydraulic due to the power, force, and frequency response requirements [5].

Active suspension exists to improve the ride and comfort over the conventional system. For instance, in a situation when a vehicle with conventional suspension system runs over a small obstruction on an even road, the wheel spring is compressed. The result is a higher wheel load and vehicle body rise. A car with active suspension can detect the wheel tendency to rise and react to adjust the actuator, maintaining constant wheel load and hence exerting no spring resistance to the obstruction. Its operation is simply equivalent to lifting the wheels to cross the obstruction [9].

The general electronic components are similar to those of the semi-active systems, with a centralised ECU and a number of sensors. There are several active suspension systems on the market. They are described by [4-6, 10-12]. One of the latest system layouts, and input and output parameters are as shown in Figure 2.9 and 2.10 respectively [12].

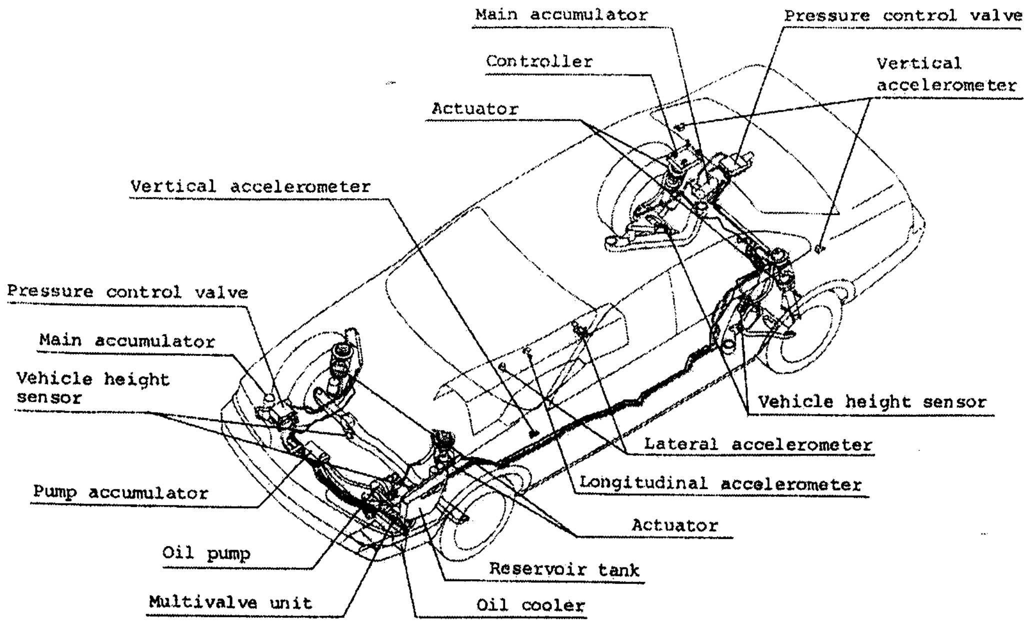


Figure 2.9 Active suspension system layout

The important vehicle ride information is provided by accelerometers that detect the vehicle acceleration in three axes, longitudinal, lateral and vertical. This information is utilised in the system and shown in the acceleration control block diagram of Figure 2.11. Its control details are described in Chapter 3.

The ECU employed in this active suspension system is capable of completing a computation cycle within several milliseconds. Such high response speed is required to suppress transient roll motion. Hence a similar update rate is required for the sensor signals.

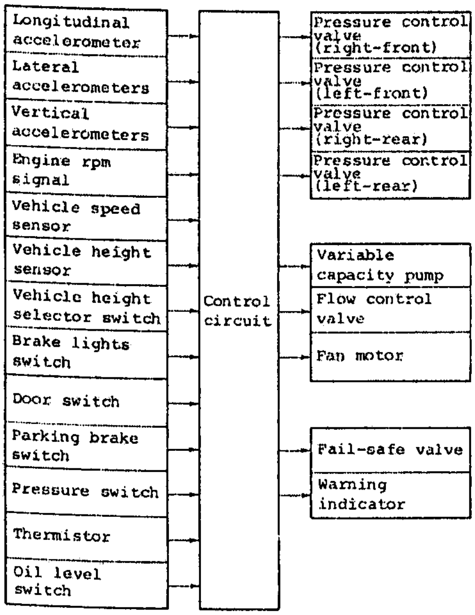


Figure 2.10 Active suspension system input and output signals

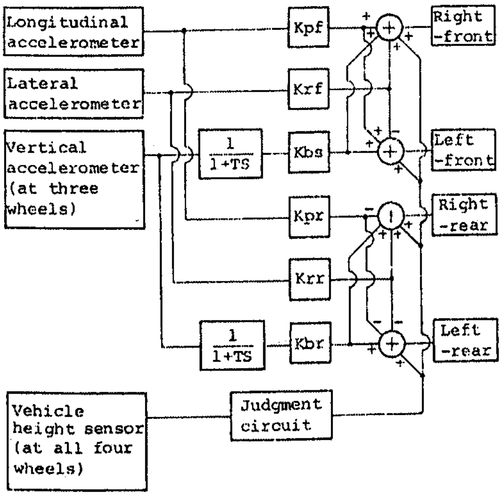


Figure 2.11 Acceleration control block diagram of active suspension

2.1.4 Suspension System Electronic Control Units

8 and 16 bit microcontrollers have been employed in suspension system ECUs. The former is found in a Rover self levelling system [15], while two of the latter are found in a more complex active suspension system [6]. One microcontroller handles data from accelerometers and issues actuator control commands, with other data, such as vehicle height and speed, processed and supplied by the other microcontroller. For system safety, each microcontroller can signal the failsafe circuit should a fault occur.

There is no information on microcontroller type used in other kinds of suspension systems. Their microcontrollers should, however, fall into either of the bit categories above, due to their lower complexity than the active suspension. On the other hand, the most complex suspension control algorithms may require a state of the art microcontroller, according to [18]. Hence it can be expected that later generations of active suspension will require a 32 bit microcontroller.

2.1.5 Current and Trend in Suspension System Research

There are currently several areas of research into suspension system. Much of the academic research goes into novel suspension control algorithms [18,98-100]. Other research is directed towards the combined control of suspension, mostly active, and other dynamic control systems, cheaper actuators and power sources [9-10,18,101-102]. The latter two are viewed as an important challenge particularly to active suspension development, since they are the most expensive part of the system and active suspension consumes significantly higher amount of energy than other types [18,97].

From the above discussion on types of suspension, it can be seen that active suspension offers the best performance. However, its high price, weight and power consumption prevent it from appearing in large volume. [97] suggests that the likely system, that could be mass produced and have performance close to that of active suspension, would be a combination of a few subsystems. The suggested combination consists of self levelling system, adaptive damping and anti-roll control. The three systems would provide a height control for payload compensation, ride control for different driving situation, and level cornering, respectively. However, despite in small volume, active suspension has been implemented on vehicles [9,11]. This proves its practicality. If there is success on its actuators and power sources research mentioned above, it could yet emerge into a higher volume car market. For these reasons, the active suspension has been chosen for the later system simulation and modelling work.

2.2 Four Wheel Steering (4WS)

Many 4WS systems have two basic operations [6]. Firstly to turn the rear wheels through relatively small angles in the same direction as the front wheels at high speed to reduce turning motion, and hence increase stability. Some cheaper systems only contain this function [6,22]. Secondly to steer the rear wheels in the opposite direction to the front wheels at low and medium speeds to decrease turning radius and increase steering response respectively. The 4WS steering concept can be illustrated, by the steering characteristic diagram of a typical system in Figure 2.12 [6].

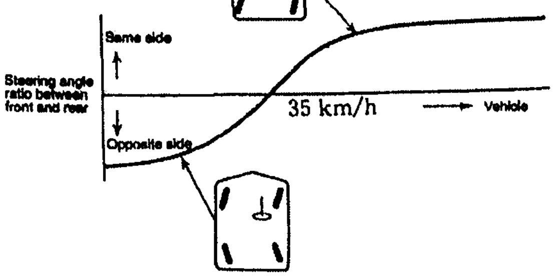


Figure 2.12 A typical front and rear wheel steering characteristic of a 4WS

2.2.1 Types of 4WS Systems

4WS systems can be classified according to their mechanism as follows [6]:

- fully mechanical with steering shaft connecting front and rear wheel steering gearboxes
- electronic-hydraulic which applies hydraulic power to steer both the front and rear wheels. The example of such system is described in [19,31].
- electronic-hydraulic-mechanical systems. One particular system has a mechanical-hydraulic steering mechanism when steering rear wheels in the opposite to the front wheels, and applies electronic-hydraulic type when steering the rear wheels in the same direction as the front wheels [6]. The two mechanisms are determined and controlled by the ECU. The system is illustrated in [20].
- electronic-electric control systems. As the name suggests, the steering of this type of system is powered by electric motor and controlled by an ECU.

The control algorithms running within these ECUs are discussed in the next section.

2.2.2 4WS Control Algorithms

Every type of four wheel steering system described above (except the fully mechanical ones), is controlled by a central ECU. The ECU processes information from sensors to determine the rear wheel steering angles and issues commands to the front and rear wheel actuators. The control algorithm running on the 4WS ECU can be of either open or closed loop type.

The open loop control algorithm applied is based on a predetermined relationship between vehicle speed, front wheel angle, lateral acceleration and steering wheel rotational velocity.

An example of a rear wheel steering characteristic of a simple system, installed on a Nissan car in 1986 is as shown in Figure 2.13 [6]. This 4WS system is intended to improve stability at high speed driving only, not to reduce the turning radius at low speed. The rear steering angle range is thus small. In this algorithm, only vehicle speed and lateral acceleration are used in determining the rear steering angle.

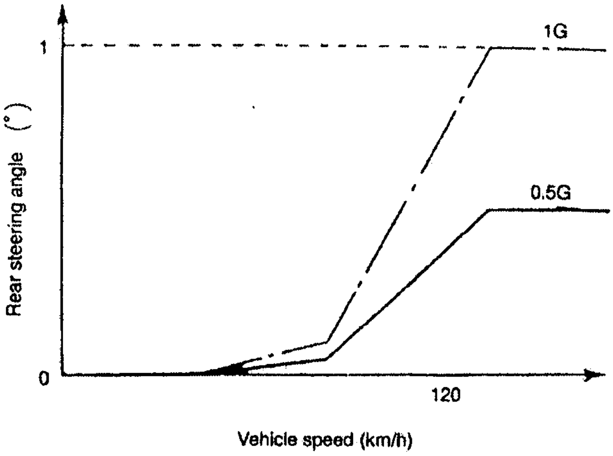


Figure 2.13 Rear wheel steering characteristics of a simple 4WS system

Recent 4WS systems use a more complicated closed loop control, using yaw rate feedback to improve stability [18]. A proportional control algorithm using steering angle, vehicle speed and yaw rate sensors is shown in Figure 2.14 [28]. The system configuration is displayed in Figure 2.15 [20].

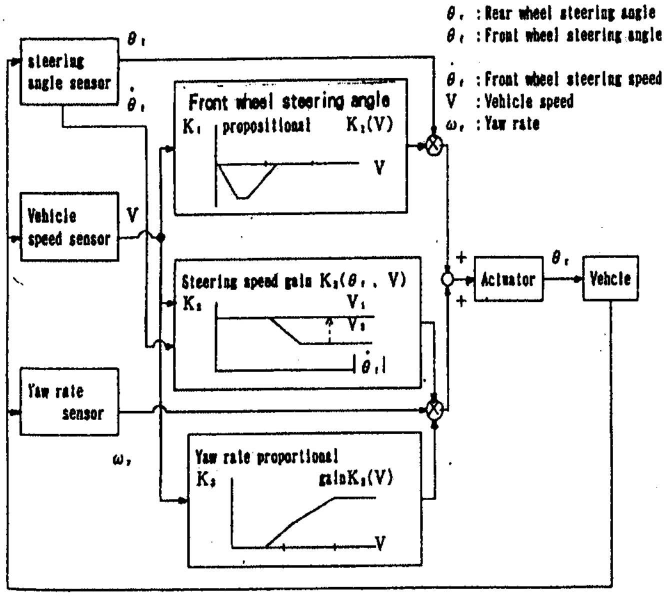


Figure 2.14 Proportional control block diagram of a 4WS system

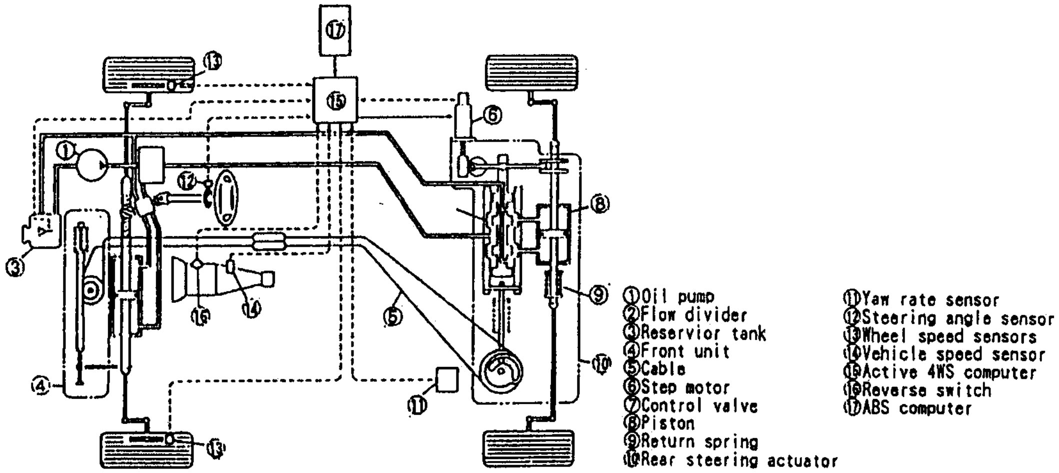


Figure 2.15 4WS with yaw rate feedback control system diagram

The control equation (2.1) is shown below [20].

$$\theta_r = K_1(V).\theta_f + K_2(\dot{\theta}_f, V).K_3(V).\omega_y \tag{2.1}$$

$\dot{\theta}_f$	: front wheel steering angle velocity
θr	: rear wheel steering angle
θf	: front wheel steering angle
V	: vehicle speed
ω_y	: yaw rate
K_1	: opposite direction steering angle proportional gain
$K_2(\dot{\theta}_f, V)$	: tuning gain of steering velocity
$K_3(V)$	: yaw rate proportional gain

The first term represents the steering angle proportional gain. It is negative at low speed, so that the rear wheels are steered in the opposite direction to the front ones at low speed. This is for turning radius reduction. It is set to zero at high speed driving. The second term is the yaw rate proportional term, which becomes dominant at high speed. The two gains are positive, indicating the front and rear wheels are turned in the same direction. The rear wheels are steered to reduce yaw rate that occurs due to cross wind or road irregularity.

2.2.3 4WS Electronic Control Units

Both 8 bit [19] and 16 bit [29-30] microcontrollers have been employed in different 4WS ECUs. These systems have similar number of system sensor inputs, though one is an experimental system [30], where a microcontroller with spare capacity may be selected to allow for subsequent program changes. In one case, two microcontrollers are fitted as a safety precaution [29]. The two processors monitor each other's control operations. If a discrepancy is detected, each is capable of switching the system to a safe state [29]. Figure 2.16 shows the ECU structure diagram.

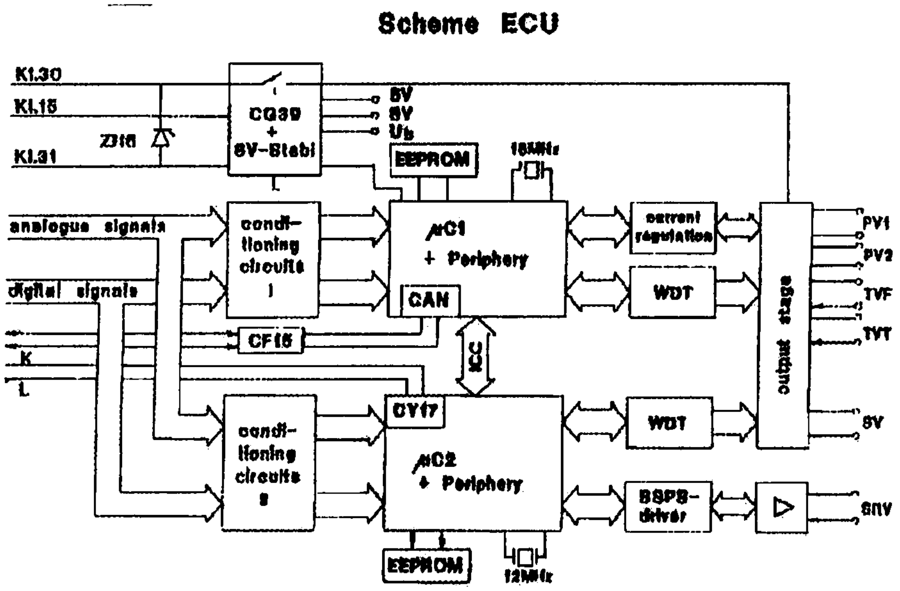


Figure 2.16 A 4WS ECU structure

2.2.4 Status and Future Trend of 4WS

Work is currently underway to introduce new control concepts to further improve system handling characteristics [6]. In the existing systems described above, the control is based on a predetermined relationship between vehicle speed, front wheel angle, lateral acceleration and steering wheel velocity. Newer concepts include control algorithms based on vehicle side slip angle and model vehicle behaviour. The former is to keep the side slip, which is the difference between the vehicle heading direction and the moving direction of the vehicle centre of gravity, close to zero for ease of driver's control. The latter acts to try and match vehicle behaviour to the desirable vehicle behaviour model, based on driver's steering operations. This new concept may also have an effect on increased system complexity.

Unlike ABS, the 4WS system has not been widely installed in modern cars, since its first introduction to the production cars in 1985. This is due to the extra cost on a vehicle, and the fact that its effect can only be realised in particular driving conditions such as obstacle avoidance [23]. Under normal driving, the effect of 4WS is almost equal to ordinary front wheel steering [23]. However, the combined effect of 4WS, suspension and ABS is believed to greatly enhance the performance and stability of a vehicle [21,24,25]. It is also considered that this combination of systems forms part of the Advanced Vehicle Control System (AVCS), which is a research topic for future vehicle dynamic control systems [26,27]. Furthermore, an attempt to introduce cheaper version of 4WS aiming for higher market acceptance is also underway [22]. For the above reasons, the inclusion of 4WS in this work is considered justifiable.

2.3 Electronics in Braking and Anti-lock Braking Systems (ABS)

The braking system is an indispensable part of a vehicle, as it is a safety-related system [35]. For this reason, there has been a large amount of research into the area. To improve the performance in various driving situations, electronic control has been introduced to prevent wheel locking, distribute braking force amongst wheels depending on load and improve the driver's response to an emergency stop situation. This has proved so beneficial that a large proportion of modern cars contain electronics as part of their braking systems. Many future developments, such as a fully brake-by-wire system will only be made possible by the use of electronics.

This report will concentrate upon an Anti-lock Braking System (ABS). This is because it is now established and widely installed in modern cars and expected to be a standard fitting in future vehicles, with its benefits in terms of improved braking performance to drivers proven and recognised [32-34].

The objectives of a typical ABS system are listed below [6,35]:

- keeping a vehicle stable during braking
- preventing a controlled wheel from locking
- the loss of ABS must not affect the safety of standard braking systems

Some ABS systems are also designed with the following objectives [35]:

- maintaining steerability during heavy braking
- optimally utilising tyre-road friction in order to minimise braking distance
- being able to quickly adapt to tyre-road friction changes
- minimise yaw effect during braking on split-coefficient road surface
- stable braking while cornering

2.3.1 Types of ABS

ABS can be categorised into three groups according to the number of braking circuits (or channels) that its hydraulic system regulates [36].

2.3.1.1 Two-channel ABS

This is the simplest and cheapest type of ABS. A front and its diagonal rear wheel are controlled together in a pair. Each front wheel is monitored for a sign of lock up. When front wheel lock up is detected, the brake pressure of both wheels in a pair are released simultaneously. In this ABS system, only two front wheel speed sensors are needed. The amount of control task is less, and hence its lower cost. The performance is inferior to the other two types of ABS. It is mostly installed in small cars.

2.3.1.2 Three-channel ABS

As the name suggests, there are three control targets in this system. The two front wheels are controlled individually, while the two rear wheels are controlled in a pair. A popular control concept for rear wheels is 'select low', which provides equal brake force to both rear wheels at all time. The amount of brake force given depends on the status of a rear wheel with lower coefficient of friction. Despite the higher cost then the previous system, its performance is considerably superior [37]. Three-channel ABS is widely employed in medium and large cars.

2.3.1.3 Four-channel ABS

This is the most costly and complicated ABS of the three types. All four wheels are monitored and controlled individually. As a result, the available tyre-road friction of each wheel is optimally utilised and hence the shortest braking distance is obtained.

2.3.2 ABS Control Algorithms

Having been in the market for more than twenty years, there has been a large amount of research into ABS control algorithms. Different control methods such as deceleration threshold, proportional-integral control and fuzzy logic are developed

[38-41]. However, only the former method is focused upon here, since most ABS systems in the market employ this deceleration and wheel slip threshold algorithm [40].

2.3.2.1 Wheel Slip

When a vehicle is accelerating or braking, tyres deform which causes the difference between a circumferential wheel speed and a vehicle velocity. The wheel is said to 'slip'. As the result, a frictional force between tyre and road surface is generated.

Slip can hence be defied in equation 2.2.

$$\lambda = (V_v - V_r) / V_v \tag{2.2}$$

λ - wheel slip
V_v - vehicle velocity
V_r - circumferential wheel speed

Figure 2.17 shows a typical relationship between slip and tyre-road coefficient of friction μ.

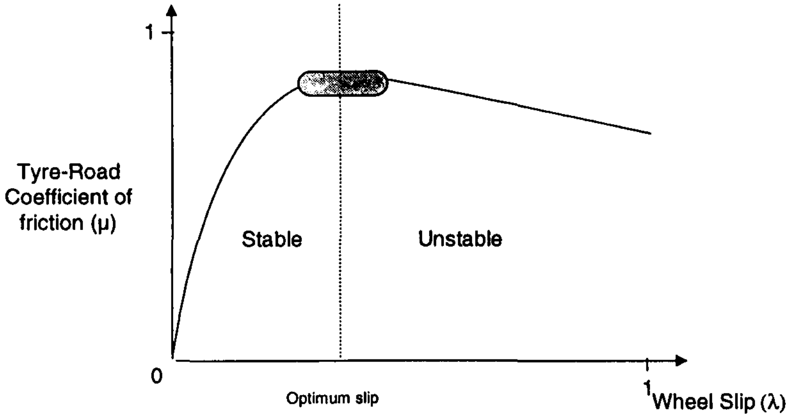


Figure 2.17 Relationship between slip and tyre-road coefficient of friction

As seen from the graph, friction initially builds up to its peak value as the slip grows. This is a stable area, where braking effort and tyre-road adhesion are balanced. Here the higher braking effort asserted, the more tyre-road frictional torque generated from the increase tyre-road friction, hence greater braking effect. Once the graph moves into a higher slip area beyond the peak tyre-road coefficient of friction, a wheel begins to lock and the balance between braking effort and friction is lost, hence less braking effect. This is an unstable area. At its maximum slip value of 1.0, the wheel is completely locked.

A basic control objective of an ABS is to prevent an excessive wheel slip beyond the peak friction coefficient into the unstable area. Its control range is designated by a shaded part of the line.

2.3.2.2 Deceleration Threshold Control

This control method is applied in ABS systems by Bosch which is widely popular in ABS market [36]. The basic control procedure is that the system continuously monitored a target wheel to see if wheel deceleration and wheel slip values exceed set thresholds. When both thresholds are surpassed, ABS will activate to avert wheel lock.

A simplified model of an ABS system is as shown in Figure 2.18 [42].

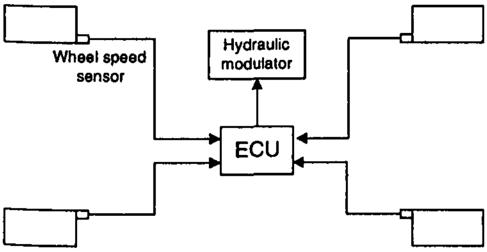


Figure 2.18 Simplified model of ABS electronic control applied to some three and four-channel systems

From the wheel speed sensors, the ECU can calculate the following control variables [43]:

vehicle reference speed — obtained from a pair of diagonal wheels such as left front and right rear wheels. During braking, its value is generated based on an interpolation of the speed at the beginning of braking process.

wheel acceleration — calculated directly as a differentiation from the wheel speed.

wheel slip — cannot be directly measured but its representative value can be calculated based on the reference speed.

Typical ABS control cycles on high tyre-road friction coefficient road can be as shown in Figure 2.19 [42].

v_F Vehicle speed, v_{Ref} Reference speed, v_R Peripheral wheel speed, λ_1 Slip switching threshold, $+A, +a$ Thresholds of peripheral wheel acceleration, $-a$ Threshold of peripheral wheel deceleration, $-\Delta p_{ab}$ Brake-pressure decrease.

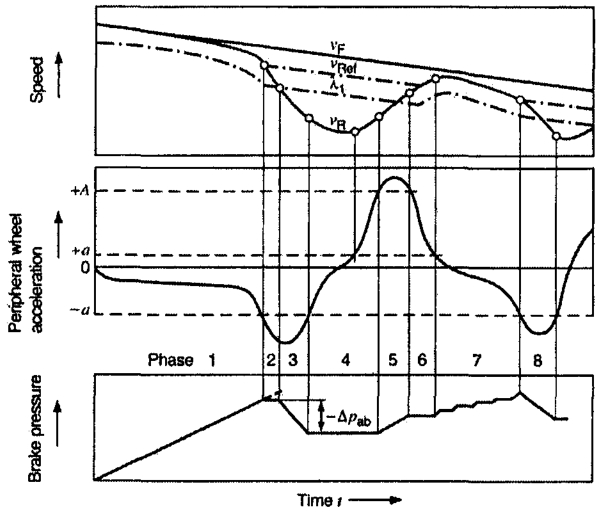


Figure 2.19 Typical ABS control cycles on high tyre-road friction coefficient road

The explanations for each control phase are described as follows:

Phase1 the vehicle is under braking, wheel and vehicle speeds reduce. At the end of the phase, wheel deceleration reaches its threshold.

Phase2 wheel deceleration threshold is exceeded, ABS holds brake pressure constant. At the end of the phase, wheel speed approaches slip limit.

Phase3 both wheel deceleration and slip exceed their limits. ABS decreases brake pressure, to let wheel gain speed until wheel acceleration goes above threshold.

Phase4 once the wheel accelerates above threshold, brake pressure is maintained. The acceleration continues

Phase5 wheel acceleration is beyond the upper limit, which signals ABS to increase brake pressure

Phase6 wheel acceleration is now below the upper limit. ABS maintains constant brake pressure, as wheel acceleration is above +a threshold indicating that it is still slightly under-braked.

Phase7 ABS builds up brake pressure until wheel deceleration threshold is exceeded

Phase8 the cycle repeats by reducing brake pressure irrespective of the state of wheel slip

2.3.3 ABS Electronic Control Units

An example of a simplified ABS ECU is displayed in Figure 2.20 [6,37,42].

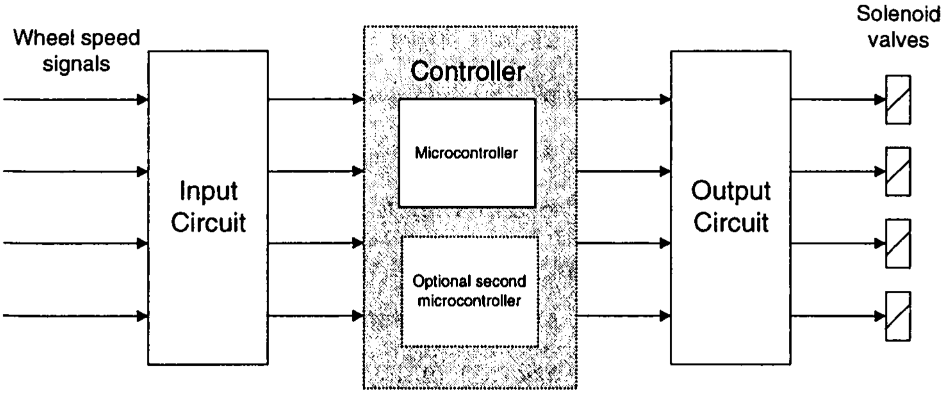


Figure 2.20 Simplified model of an ABS ECU

The input circuit contains a filter and an amplifier. Its functions are filtering input noise, amplifying the input signal, and converting it into a form ready to be processed by a microcontroller.

The controller is a heart of all the control. It executes the control algorithm, by calculating controlled variables from wheel speed signals, and processing them according to the control rules. It finally sends command signals to drive circuits and warning lamp.

The output circuit receives control signals. It then regulates and amplifies current for driving the brake solenoids.

2.3.3.1 Microcontrollers for ABS

There is a wide variety of microcontrollers used in ABS systems, dependent on the system complexity. Both 8 and 16 bit microcontrollers can be seen to satisfactorily perform the control tasks of a modern practical ABS [36,44-48]. [36] suggests that an 8 bit microcontroller is a cost-effective choice for a simple two-channel ABS, whereas a more complex four-channel system demands a higher performance 16 bit microcontroller. Due to the real time nature of the tasks, the microcontroller needs to be able to execute them reliably within a fixed interval. It has to calculate wheel speed at the typical cycle speed of 5ms [36,46]. Some additional hardware such as timer capture input can help a 8 bit microcontroller achieve this target [44].

Some ABS systems employ a combination of microcontrollers in their ECUs to meet performance and cost targets. An example is the application of three 8 bit microcontrollers, to carry out distributed calculation tasks of a combined ABS and traction ECU [45]. Another system employs two microcontrollers, each being responsible for the control of two channels of a four-channel ABS system [42].

Another important reason of using extra microcontrollers in an ABS system is for safety purposes. One system has a high performance 16 bit microcontroller for its control tasks, while engaging a medium performance 8 bit microcontroller as a safety computer to check the correctness and timing of commands [47]. The idea of having a second identical microcontroller to perform duplicated control calculations is also popular [37,49]. In these systems, both microcontrollers receive the same wheel speed inputs and execute the same control algorithm to obtain output signals. The pair monitors each other's output signals. If the two are different, indicating an error, the ABS is shut down and a warning lamp is lit to warn the driver [49].

2.3.4 Current Research on ABS

The current research on ABS can be summarised as having the following objectives [50]:

- to improve system and assembly design technique to reduce cost, so the ABS is more suitable for medium and low price vehicles. An example of area which needs addressing is the amount of electrical connectors used for wiring, which has an increasing percentage of overall system costs [50].
- to improve the utilisation of braking traction available on a wider range of road conditions, to increase the system performance
- to integrate the system with others, such as traction control and electronic brake distribution, to improve vehicle handling characteristics
- to enhance communication with the driver and compensate for driver incapability, for example, automatic braking to avoid a crash due to slow human reaction.

Having gone through the reviews of the three dynamic control systems, the microcontroller, which is the central part of those systems, will be reviewed.

2.4 Microcontrollers

2.4.1 Introduction

The microcontroller is a specialised microprocessor modified to suit it to control applications [51]. It is employed in various specific embedded control applications, such as in electronic appliances, robot arms, vehicle electronic circuits, etc. Due to the variety of its applications, there is a wide range of peripheral components that can be included in the microcontroller to suit each specific application, such as Analogue-to-Digital Converter (ADC). Also there is a large number of possible specifications for a microcontroller for each control task. These specifications such as memory size, number of Input/Output (I/O) ports, processor speed, and data word length (bits) are the basis for a designer to select a microcontroller for an application.

2.4.1.1 Difference Between Microprocessor and Microcontroller

The two devices are generally used in different applications. The microprocessor is used in computer applications, whereas the microcontroller is applied in control and instrumentation. Their functional requirements thus differ.

The microcontroller may have to monitor a large number of input sensors, and send output signals to various actuators at the same time. Therefore, it usually possesses more input/output (I/O) ports. Its design objective is also towards high performance and speed in I/O operation, rather than calculation-intensive applications. Control programs are usually smaller than computation programs, and less memory space is needed. Hence a smaller Read Only Memory (ROM) section, where control program is stored, is contained in the microcontroller, compared to the larger size in microprocessor. Microcontrollers, therefore, mostly have built-in memory which is sufficient to use in their standalone control applications.

For the same reason, a microcontroller also incorporates a timer and some necessary components for a specific control task [56]. Examples of such functions are the analogue to digital (A/D) converter, digital to analogue (D/A) converter and multiplexer. Its clock speed is also generally slower than that of the equivalent microprocessor.

Due to the severe environment in some embedded control applications, such as in automotive use, microcontrollers are designed to be able to withstand higher operating temperature than microprocessors, of which the applications are mostly in an office environment.

2.4.2 Architectures

The standard architecture of a microcontroller is as shown in Figure 2.21 [6].

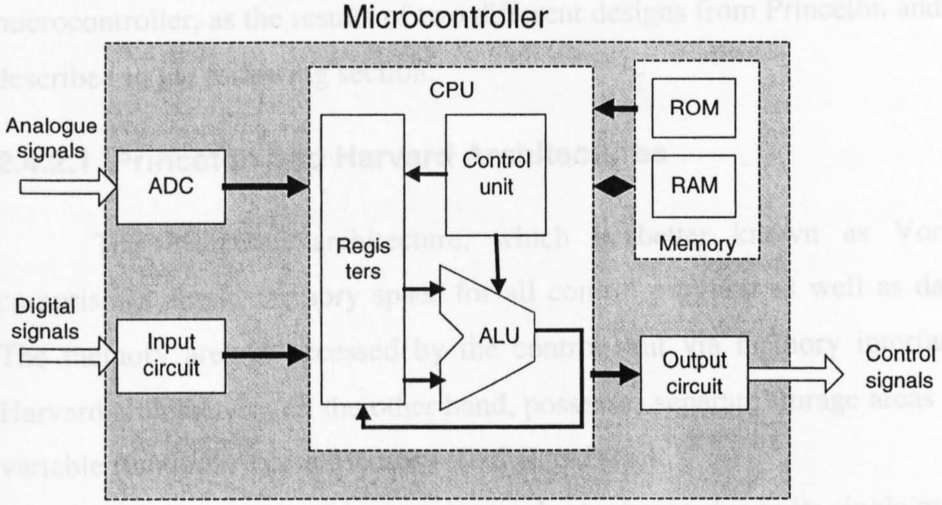


Figure 2.21 Microcontroller simple block diagram

A microcontroller receives input signals from an external source via ADC and digital input circuits. During operation, the microcontroller takes commands or instructions from a control program stored in ROM.

The control unit, as the name suggests, controls the whole operation of the microprocessor according to the control program by directing other parts of the microcontroller. It controls data access from memory and tells the Arithmetic Logic Unit (ALU) to perform specified logic and arithmetic operations.

The microcontroller also contains different kinds of registers [51]. The general purpose ones are:

- program counter which stores the address of the next instruction in the memory, to be retrieved.
- instruction register which keeps the instruction code, that is next to be executed.
- accumulator which contains the operand or data, to be performed arithmetic or logic operation.

- address register which contains the memory address of data to be retrieved or stored.

Additionally, specialist registers may be used such as reset source register which indicates the sources of the latest microcontroller reset [103], and time base counter which increments its content at a variably set frequency to provide time base for a program [104].

There are, however, two alternative ways of arranging memory in a microcontroller, as the results of two different designs from Princeton and Harvard, as described in the following section.

2.4.2.1 Princeton and Harvard Architectures

The Princeton architecture, which is better known as Von Neumann, comprises a single memory space for all control program as well as data variables. The memory area is accessed by the control unit via memory interface unit. The Harvard architecture, on the other hand, possesses separate storage areas for program, variable Random Access Memory (RAM) and stack.

The Princeton architecture is simpler to design due to its single memory and is more flexible in developing a software, whereas the Harvard architecture tends to require fewer clock cycles to execute an instruction code [52]. The two most popular 8-bit microcontrollers, Intel 8051 and Motorola 68HC05, employ different architectures. The former is based on Harvard, while the latter applies Princeton architecture. However, a physically larger and more complex Harvard architecture is the architecture for most modern high capability microcontrollers because of its powerful support for parallel fetching of data and instructions [67].

In terms of microcontroller instruction sets, microcontrollers can be divided into Complex Instruction Set Computer (CISC) and Reduced Instruction Set Computer (RISC).

2.4.2.2 CISC and RISC

The two terms are not clearly distinguished and some microcontrollers have characteristics in between the two. The majority of microcontrollers are CISC, which tend to have a large number of instruction sets, some of which perform variation of the same operation e.g. load data directly, load data by indirect addressing. The concept of RISC is to have fewer numbers of simple instruction sets, and leave more complex operations software design to users. The RISC concept is a microcontroller designed to perform all instructions in single clock cycles, though this is not the case for some RISC devices [60]. Another difference is that the RISC processor does not have microcode memory, which acts as a decoder for software instructions in CISC. RISC instructions are fed directly to a logic device, which generates signals for a control unit to execute the instructions. This results in more complex chip design but the omission of microcode memory enables smaller silicon size and generally faster speed [60].

The performance of the two architectures are program dependent and, therefore, one cannot be judged to be better than the other in all applications. The current design trend is towards RISC [53]. This trend also applies in the automotive industry [60]. This is especially true in the application of powerful 32-bit RISC microcontrollers used in powertrain control [57,60].

2.4.2.3 Memory

Microcontrollers can use internal or external memories. Because of their generally severe operating environments, both types of memories are served by compact and temperature tolerant semiconductor memory devices.

These can then be divided into two classes [54]: read only memory (ROM) for non-volatile program storage, and read/write memory (RWM) or a more common term of Random Access Memory (RAM) generally for variable storage.

The established types of memory in automotive systems are ROM, Erasable and Programmable ROM (EPROM) and flash memory, for program storage [70]. RAM is used for stack and variable storage, while byte erasable EEPROM (Electrically Erasable and Programmable ROM) is installed with calibration and security data [70].

Flash memory is sometimes called flash EEPROM due to the similarity to EEPROM in a way that it can be electronically erased and programmed. The two names are interchangeable as their definitions are still confusing [52]. One source says they differ in programming mechanism [6], whereas another suggests that EEPROM can be partly erased while flash content can only be deleted in bulk [59,60]. Flash and EEPROM offer the useful opportunity for an on board programming and easy aftersale software update and revision. Now flash memory is becoming less costly and hence expected to ultimately replace ROM as the program memory storage [59,70].

There is currently limitation on the amount of program memory available on a microcontroller, due to its relatively large silicon space required. A Motorola 68HC916Y5 microcontroller with 164 Kbytes of flash ROM, which is among the largest embedded memory size, has half of its entire space occupied by memory [111]. The chip space corresponds to its cost, so this emphasises the need for good system design to minimise memory usage and hence its production cost [112].

RAM requires 10-15 times more space and is 10-15 times more expensive per byte than ROM [112]. However, less RAM is generally required for embedded control applications than ROM. The estimate RAM usage has been given as 12-20 times and 32 times less than ROM by [113] and [70], respectively. Modern automotive microcontrollers contain up to 8 Kbytes of RAM [114].

2.4.2.4 Processor Speed

The speed of a processor, which implies an ability to perform control tasks or information handling, depends on many factors.

The number of clock cycles required to perform an operation also directly affects the speed of the microprocessor. With the same clock frequency, a microprocessor, which requires 4 clock cycles to perform a specific operation will be 20% faster than the one, which needs 5 clock cycles. Its range of instructions also has an effect, for examples, a microcontroller without division instruction generally needs a long and repetitive program to divide numbers, and hence taking relatively long time to perform this particular task.

Modern microprocessors have master clock speed as high as 1.5 GHz [69]. Most of them, nonetheless, require several clock cycles to execute one operation. Even the fast model such as Motorola 68040 still has an average of 1.25 cycles per operation [55]. Unlike, microprocessors, Microcontrollers run at a much lower operating speed, with some fastest devices operating at 40 MHz [67]. This is mainly due to the heat, ElectroMagnetic Interference (EMI), and power consumption constraints.

Microprocessor speed can also be slowed down by multiplexed address or data bus, as well as the input/output (I/O) operation with external devices.

Designers also need to take power consumption, heat, and EMI into consideration, as a vehicle is a tight space, with high concentration of electronics and of limited power supply environment. It is generally accepted that higher microcontroller operating speed implies higher heat generation, power consumption and also EMI. The use of minimum clock speeds possible is recommended in system design practice [68].

2.4.2.5 Interrupts

The interrupt is an essential part of real-time control, to allow interaction with external signals which need urgent action. Wheel speed and engine speed are examples of signals that are usually handled by interrupts. In order to service an interrupt, the microcontroller stops executing current program and performs the pre-programmed interrupt tasks. Most modern microcontrollers have a number of interrupt levels, associated with the sources of interrupts, for instance Input Capture ports or CAN port [63-65]. In case more than one interrupt occurring at the same time, the interrupt with higher priority will get serviced first. Some advanced microcontrollers also contain hardware dedicated for interrupt handling, which is capable of saving register data, managing A/D conversions and generating Pulse Width Modulation (PWM) signals. This hardware can execute an interrupt faster than the controller CPU itself [63].

2.4.2.6 Timers

Timers are also essential for real-time operation of a microcontroller. Applications of timers, together with other devices such as High Speed I/O (HSIO), include operation monitoring, output compare, input capture and pulse accumulation.

Operation monitoring is carried out by a watchdog timer, which continuously counts up or down. If it is not reset by the program code before the count reaches a certain set value, a hardware reset signal is generated to re-start the microcontroller. This is to prevent ECU software from getting stuck in an infinite loop.

Output compare is applicable to generating output signals such as square waves, or Pulse Width Modulation (PWM) output, which require precise period or duty cycle provided by timers. This function of timers also includes time delays generation without CPU burden.

The Input capture function is done by sensing transitions of an input signal and recording their times. This enables a microcontroller to measure the pulse width of an input signal. A wheel speed signal is captured and its frequency, which indicates wheel speed, is measured in this manner.

Pulse accumulation involves the counting of the number of times an event such as input rising edges occurs.

Modern microcontrollers now contain several timers to cope with intensive real-time control nature of automotive applications [63-66]. Hardware peripherals to help ease CPU burden in performing aforementioned functions are also sometimes included. Dedicated controllers for timer related operations, which effectively remove CPU overhead on timer control, are also introduced on some recent versions of microcontrollers [63,66].

2.4.2.7 Inputs/Outputs (I/O)

I/O is critical to the operation of a microcontroller, as its tasks require frequent contact with external devices. High speed I/O, low speed I/O and serial I/O are the three types of I/O available in modern microcontrollers. Some are capable of configuring their I/O ports to switch between these functions [6].

High Speed I/O (HSIO) is interrupt driven, and its functions are often associated with timers in input capture and output signal generation. These functions are described in the previous section.

Low Speed I/O (LSIO) is a means of parallel data transmission between external devices and the microcontroller. Instead of interrupt driven control, LSIO data reception and transmission are manually control via a dedicated register, by the user program.

Serial I/O is developed for data transfer with minimum microcontroller pin requirement. As the name suggests, data is transferred bit by bit i.e. serially through a single medium. The data transmission can be either synchronous or asynchronous. A synchronous transmission requires clock signal link between the microcontroller and a communicating device, so that each bit of data is sent on each clock transition. Despite several variations, asynchronous transmission includes sending a start and end bit before and after the data is transmitted, respectively, to notify the receiver.

As mentioned earlier, modern microcontrollers contain dedicated auxiliary processors to perform I/O related to interrupts and timers [63,65]. This is to help free more of the main CPU time. Some peripheral pins of later microcontrollers can also be configured to be employed in I/O task [66].

2.4.2.8 Analogue-to-Digital Converter (ADC)

ADC is included in a microcontroller to convert analogue input signals into digital forms for control use. The most popular integrated ADC are of the Successive Approximation type, due to its best compromise between accuracy, conversion speed, and chip space requirements. The method is to compare the input signal with a series of analogue reference voltages until the closest match is found. The reference voltage is adjusted by the factor of 2 for each test. Starting with the voltage half of the conversion range, if the input is larger or smaller than the mid range voltage, the reference voltage set for the next test will be ¾ or ¼ of the conversion voltage range, respectively. The process continues until the closest match reference voltage is found. For a 8-bit resolution conversion, a maximum of 8 comparisons are required to complete the conversion. Current microcontrollers contain 8, 10 or 12 bit ADC [105].

In order to handle multiple analogue inputs with a single ADC, it is usual for the microcontroller to multiplex up to 16 channels of analogue input to a single ADC.

2.4.3 Microcontroller Trends in Automotive Electronics

The trend of employing more microcontrollers in cars is prominent, as shown by the prediction in Figure 2.22 [70]. The prediction agrees with a number of predicted drastic rises of electronic costs and cost percentage to overall vehicle costs, by authors in semiconductor and automotive industries [84,106-110]. This is due to the number of new electronic control systems which have been anticipated in future vehicles, such as satellite navigation and crash avoidance systems.

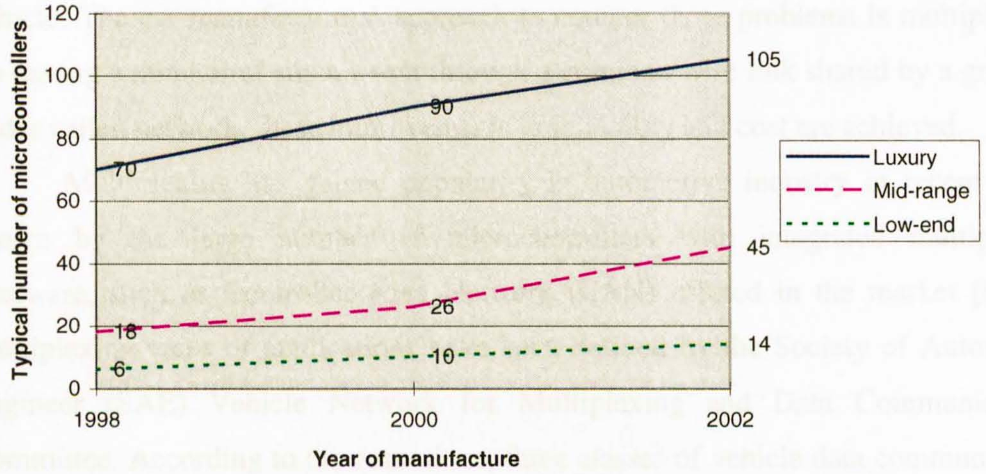


Figure 2.22 Microcontroller implementation growth in modern vehicles

There is also a trend of applying more powerful microcontrollers to existing systems, as automotive electronic control tasks become more complicated, leading to 16 bit microcontrollers appearing in place of the most common 8 bit devices [58].

The other reason for using more powerful microcontrollers is the more advanced diagnostic requirements now appearing, due to stricter regulations and higher reliability demands from customers. Diagnostic tasks are believed to take as much as 50% of microcontroller performance in some applications [61] and can also occupy the same ratio of program code [62].

The widespread adoption of networking in vehicles also yields the need for microcontrollers with networking capability e.g. the ones with Controller Area Network (CAN) interfaces. These types of microcontrollers are now widespread in the market, and will be discussed further in the following section.

2.5 Multiplexing and Controller Area Networks (CAN)

2.5.1 Multiplexing

Multiplexing is a method of combining signals on a single circuit, of which the benefits to a vehicle are to minimise cost and improve reliability of wiring. In the past, each of the electrical signals sent within a vehicle is via an individual wire link. As the electronics in the vehicle becomes more and more complex, this gives rise to increased numbers of sensors and actuators, and also connections and wiring harness required. Furthermore, the wiring harness is the most common source of failure in automotive electronics [71]. It also significantly contributes to the overall cost of a vehicle. The car manufacturers' approach to counter these problems is multiplexing. By having a number of signals sent through a common wire link shared by a group of nodes called network, these improvements in reliability and cost are achieved.

Multiplexing has gained popularity in automotive industry in recent years, shown by the large number of microcontrollers with integrated multiplexing hardware, such as Controller Area Network (CAN) offered in the market [82,83]. Multiplexing areas of applications have been defined by the Society of Automotive Engineer (SAE) Vehicle Network for Multiplexing and Data Communications Committee. According to the committee, three classes of vehicle data communication networks are defined as [6,72]:

Class A. A potential application of multiplexing by transferring a group of signals through a common signal bus between nodes, that would have been communicated via individual wires in a conventionally wired vehicle. In that conventional vehicle, these nodes do not exist.

Class B. A potential application of multiplexing to transmit and receive data between nodes to eliminate duplicate elements such as sensors in the system. These nodes normally exist in a conventionally wired vehicle.

Class C. A potential application of multiplexing to the high speed data transfer between real-time control system modules.

Class B bus must be capable of doing all of Class A communication, and hence the superset of Class A. On the other hand, Class B is the functional subset of Class C. Class C bus must therefore be able to perform all the three class functions.

In this thesis, only Class C bus is concentrated upon, since it is the means of multiplexing real-time control systems such as engine management system, transmission system and ABS, which is of interest here. Class C is defied as having the minimum data transfer speed of 125 Kbit/s up to 1 M bit/s to ensure satisfying real-time response [77].

Among Class C multiplexing, Controller Area Network (CAN) has been the most widely used protocol and has been adopted as the high-speed networking protocol in Europe [74-75]. CAN is also gaining more acceptance in the US. Hence, it is believed that it will become a global standard system in the future. CAN is, therefore, the sole protocol considered in this thesis.

2.5.2 CAN

CAN protocol is based on CAN specification by its developer, Bosch [76]. The summarised specification is described in this section.

2.5.2.1 System Overview

An example of the high-speed CAN system is as shown in Figure 2.23. From the diagram, CAN system has a physical layer called the *CAN bus* which is the medium for data transfer. All the modules or nodes can gain access to transfer or receive data from the bus.

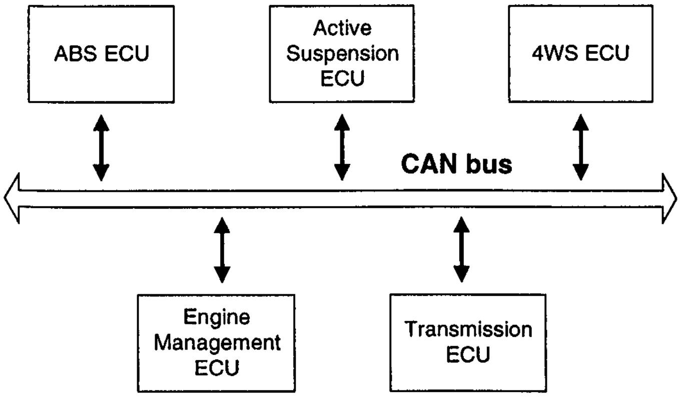


Figure 2.23 System block diagram of high-speed CAN network

2.5.2.2 Bus Configuration

All the nodes connected to the CAN bus in a CAN network are equal in terms of priority, according to the multi-master principle. The failure of one module will not affect access to the bus of others. Priority in data transmission does not depend on the module, but on the importance of the data, as part of each data has the number indicating its priority.

2.5.2.3 Message Addressing

Every message in CAN has been assigned a label or *identifier.* It is recommended that each message has a unique identifier. which gives an information on the type of message for example right rear wheel speed sensor.

2.5.2.4 Message Types

There are four different message types for transfer:

A DATA FRAME used when a node is transmitting data.

A REMOTE FRAME sent by a node requesting DATA FRAME with the same identifier.

AN ERROR FRAME transmitted when a node detects a bus error.

AN OVERLOAD FRAME used to give extra delay between DATA or REMOTE FRAMES transmission.

2.5.2.5 Message Format

There are two types of CAN DATA FRAME message formats: standard frames and extended frames. The only difference between the two are that the former has 11 bit identifier, but the latter contains a 29 bit identifier. The standard and extended frame CAN are called CAN 2.0A (formerly known as CAN 1.2) and CAN 2.0B respectively. Figure 2.24 displays a general CAN message format without length indication [76].

The arbitration field is where the identifier is contained with another control bit to indicate whether the message is DATA or REMOTE FRAME. The identifier is used to decide which message is to be transmitted in case a message collision occurs.

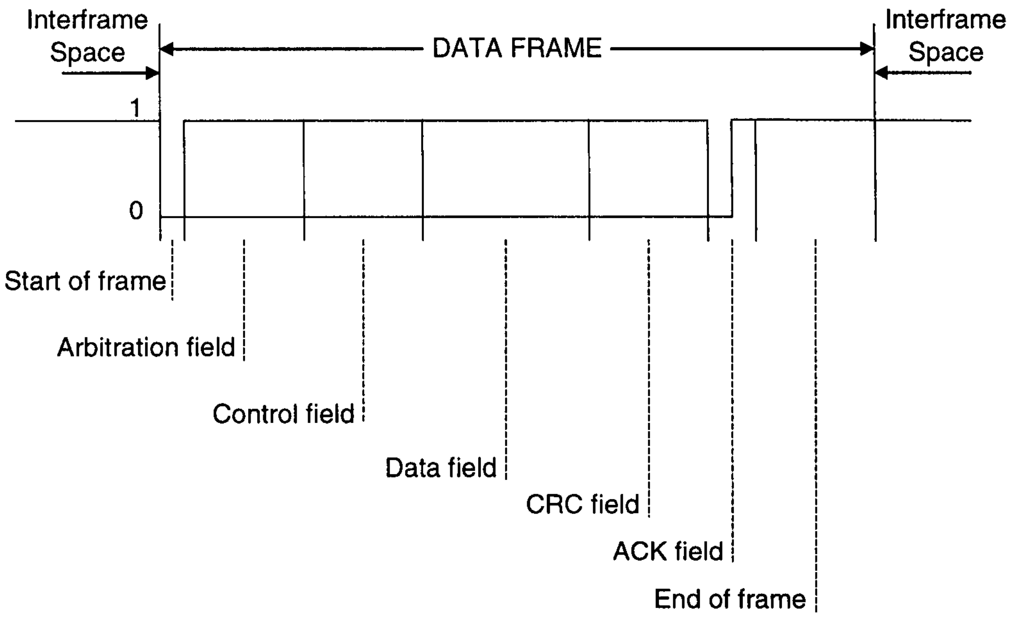


Figure 2.24 CAN DATA FRAME message format

2.5.2.6 Bus Access

Each node can transmit a message when the CAN bus is free. When more than one node is trying to transmit a message, a selection process called *non-destructive bitwise arbitration* occurs. The process monitors the identifiers of simultaneously sent messages bit by bit.

Each bit in a message is either recessive (logical 1) or dominant (logical 0). Starting from the first bits of the identifiers, during the arbitration, if the identifier bits are the same, the next bits are compared and so on until two corresponding bits are different. At that point the message with dominant bit wins and will continue to be transmitted, while the other message with recessive bit has to stop sending. It will then wait until the bus is free before attempting to transmit the message again. If there are more than two messages being sent at the same time, the arbitration works in the same way. It eliminates the messages until the only one with the longest sequence of dominant bits is left sending.

2.5.2.7 Latency Time

With CAN non-destructive bitwise arbitration, no transmission time is lost in message collision. CAN system can guarantee that the highest priority messages are transmitted with little delay. This is crucial in real-time control which contributes to the popularity of CAN in such applications. However, it cannot guarantee latency times for lower priority messages. Latency is defied as the time taken to send a message, measured from when a transmitter is ready to send the message until the message is received by the receiver. Latency includes the time taken to re-transmit the message in case of message collision.

2.5.3 CAN Microcontrollers

There exist two types of CAN in applications: basic CAN and full CAN. In basic CAN, a microcontroller checks all the messages on the bus to receive relevant ones [75,79]. This takes up considerable amount of processor time, thus limiting practical transmission rate to 250 kbit/s. A full CAN microcontroller possesses an acceptance filter, which masks out irrelevant messages, freeing a CPU to perform its control tasks [75,79]. This enables the full CAN system to have a transmission speed of up to 1 Mbit/s. Furthermore, full CAN allows both the standard (CAN 2.0A) and extended frame (CAN 2.0B) data formats.

In the semiconductor market, microcontrollers with integrated CAN peripheral and stand-alone CAN microcontrollers are offered by manufacturers. The latter is employed to a microcontroller without CAN to provide CAN capability. Before deciding which approach is to be applied in an ECU, the following points should be taken into consideration [80]:

Implementation cost This cost includes development and manufacturing costs. For hardware design, ECU with integrated CAN microcontroller simplifies a designer's tasks as it requires fewer peripheral components and connections than the stand-alone CAN microcontrollers. Time and hence cost of software design for the two types of ECUs are comparable, because both need software developed for reading and writing CAN messages. Due to its fewer number of components, the ECU with integrated CAN microcontroller is cheaper to produce. In terms of the microcontroller chips themselves, a combined price of stand-alone CAN and a control microcontroller is cheaper than that of integrated CAN microcontroller in low volume. However, it is the other way round for high volume purchase.

As a whole, an ECU with integrated CAN microcontroller has a potentially lower implementation cost than that using stand-alone microcontroller.

Design flexibility An ECU with Stand-alone microcontroller is more flexible to design than the one with integrated CAN microcontroller, due to the fact that stand-alone CAN controllers are able to interface with different microcontrollers. It is therefore possible to reuse some software from one system to another. In a type of system where new products are introduced regularly, stand-alone CAN microcontrollers yield greater flexibility in design and hardware modifications. Engine management systems, which need to be modified for newer engine range or change in regulations, or active suspension systems where technology is progressing, would benefit from this design flexibility.

CPU burden Research by Intel shows that the level of CPU burden of ECU with integrated CAN peripheral microcontroller is generally lower than that with stand-alone CAN microcontroller, with the possible burden on the latter microcontroller up to twice as much as that of the former [80]. This is because of the higher speed and efficiency of the interaction between the CPU and its on-chip CAN peripheral, as compared to an external link between a microcontroller and its stand-alone CAN chip.

System reliability From the manufacturing point of view, an ECU with integrated CAN microcontroller requires less printed circuit area, fewer connections and components than an ECU with stand-alone microcontroller. As a result, an ECU with integrated CAN microcontroller exhibits more reliability. Moreover, a microcontroller with integrated CAN is developed in one piece, whereas a stand-alone chip has to interface with a microcontroller possibly from another manufacturer. Compatibility has to be ensured in the latter case. The number of components also inversely contributes to the overall ECU reliability, as the individual defect ratio has to be summed.

From the above considerations, to select between a system with stand-alone microcontroller and the one with integrated CAN microcontroller, one has to weigh design flexibility against cost, CPU burden and reliability.

The popularity of CAN is still growing, after having been made standard for high-speed networking protocol in Europe and basis for J1939, the class C network for truck and bus applications in the United States [78]. There is now a move by

European car manufacturers and suppliers, to standardise the interface between distributed electronics or ECUs in cars, under the project called OSEK [81]. OSEK covers standards for an operating system, communication system and a network management system. Although an OSEK communication system is designed to accommodate most networking protocols, CAN is the protocol used in most cases [73]. Standard proposals have been made for the International Standards Organisation (ISO) 7-layer Open Systems Interconnect (OSI) model, for CAN to define the lower layers and OSEK to define the higher layers [74]. If OSEK is globally applied in the future, an application of CAN in automotive will be even more widespread.

2.6 Automotive Electronic Architecture

At present, the automotive electronics trend is to become increasingly more complex. Its importance in modern cars will grow, and hence the higher percentage cost of electronics equipment per vehicle, which is expected to reach 20% of the whole vehicle cost [84]. From the early days when electronics was introduced to the control of engine timing and fuel injection, to keep emission under new, more stringent laws, today they are also applied to improve ride, comfort, safety, etc.

This trend leads to the introduction of new systems such as crash avoidance and navigation systems, and the existing systems become more complicated with higher capability, for example, active suspension and four wheel steering. The consequences of these are the increasing number of peripherals such as sensors and actuators, and also larger wiring harness. This increase in wiring inevitably gives rise to vehicle weight, cost and reliability problems.

There are currently 2 major approaches to reducing this complexity [85]. In the first method, networking architectures such as CAN or Vehicle Area Network (VAN) have been developed. The basic idea of networking or multiplexing is to share communication lines for information transfer between components in a system or different systems, instead of having one separate wire for transferring each type of information. The second approach being to functionally integrate the relevant controllers [85,86], for example, the combination of engine management system and transmission control into one control unit.

Based on these two techniques, four arrangements of electronic systems for vehicle dynamics and powertrain controls are studied. These architectures including the typical centralised controller can be classified and discussed as follows:

2.6.1 Conventional Architecture

In this arrangement, different systems such as engine management or suspension controls have their separate, centralised ECUs. Each ECU is connected to its sensors and actuators via hard wire links, each of which is for one input or output signal. Some information such as sensor readings may be shared among systems by having extra wires between participating ECUs. The architecture is illustrated by Figure 2.25.

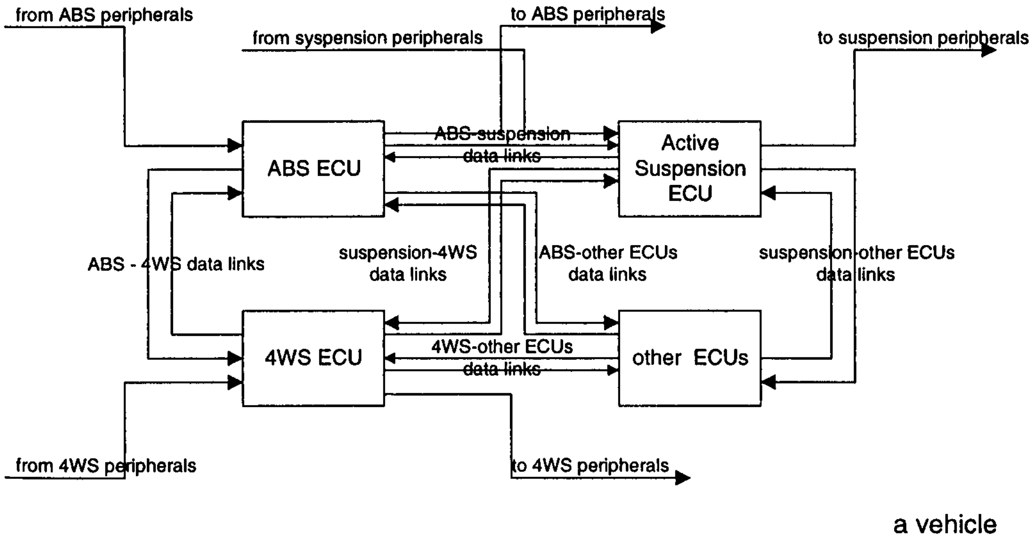

from ABS peripherals
from syspension peripherals
to ABS peripherals
to suspension peripherals
ABS ECU
ABS-suspension data links
Active Suspension ECU
ABS - 4WS data links
suspension-4WS data links
ABS-other ECUs data links
suspension-other ECUs data links
4WS ECU
4WS-other ECUs data links
other ECUs
from 4WS peripherals
to 4WS peripherals
a vehicle

Figure 2.25 Conventional centralised architecture

One of the benefits of this architecture is the ease of understanding what is contained in the vehicles, systems and their functions. In terms of development, a manufacturer can simply assign the task for each system independently to a supplier. The interconnection/communication technology required in terms of electronics is relatively low.

However, as previously mentioned, as the number of electronic systems increase, the large number of disadvantages is clearly realised. Firstly, the large number of wires causes significant rise in weight. Modern luxury cars contain a few kilometres and approximately 100 kilograms of wiring [87]. Another direct effect includes higher cost. The main wiring harness is a major bought-in component of a modern vehicle. McLaughlin et al considers it to be the second most expensive item in a vehicle, after the engine [88]. Higher fuel consumption due to weight increase also contributes to growing running costs. Manufacturing time and cost are inevitably related to the amount of wiring. Reliability problem also arises with the expanding number of wires.

Some sensor duplication may also be possible because different subsystems may be developed separately. Additionally, an introduction of a new system is relatively costly due to the lack of component sharing.

Other suggested architectures for improvement will now be presented.

2.6.2 Centralised Controller

This idea supports the integration of ECUs into one single ECU. Millward [89] claims that with the power of modern electronics, the architecture is feasible provided that the integrated systems are not too complex or including too many sub systems. For example, integrating only powertrain control systems is possible. Emaus [90], however, suggests that more systems could be integrated, as shown in Figure 2.26. The diagram also typifies this architecture arrangement.

The possible benefits discussed by Millward [89] include the probable lowest unit cost, due to the fact that it is one single unit with no duplication. Communication delay problem also vanishes, since information sharing between systems is achieved by integrating their controllers.

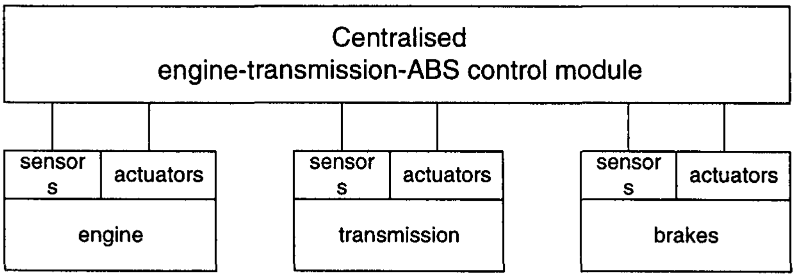


Figure 2.26 Centralised controller architecture

Referring to Figure 2.26, having had an ECU of this magnitude and complexity would raise the issues of its location and space requirement, coupled with the size of the harness needed. Installation problem is therefore imminent. Heat dissipation and Electromagnetic Compatibility (EMC) are also of concern. Another difficulty related to its complexity is the development time and resources required, including software and hardware [89]. There are also other concerns on the safety measure in case of system failure, and small amount of system sharing between different vehicle models.

The substantial effect of drawbacks over the advantages is expected to render the fully integrated single ECU, for the whole vehicle dynamical control system, impractical.

2.6.3 Peer to Peer Networking

The basic arrangement of such system vehicle dynamic control is as shown in Figure 2.27 [91]. The system consists of separate electronic control systems, which are linked by a communication network. For information sharing between different networks, there exists a data converting gateway. The gateway performs necessary conversion from one form of network message to another.

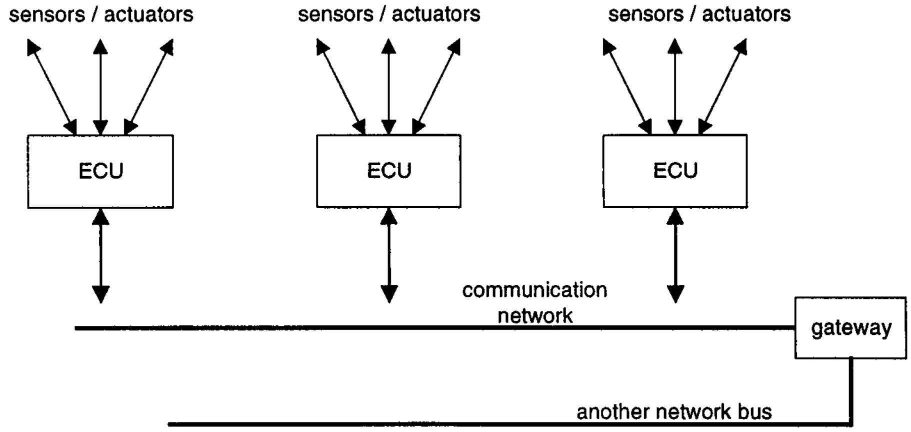


Figure 2.27 Peer to Peer network architecture

Kiencke, et al. [85] suggest that there should be local integration of frequently sharing information ECUs such as engine management and transmission control. This could reduce the number of components required by removing duplication. However, ECU integration is limited by system complexity, which would involve both technical issues, such as heat dissipation of an ECU and wiring implementation [86], as well as organisational difficulties with increased development cost and time. Therefore, the system coherence and production volume need to be taken into consideration to justify an integration.

Holzinger, et al. [86] point out that cost can be saved by integrating electronics into a related mechanical system, for example, having hydraulic modulator and ABS ECU as one piece. The saving would come from reductions in wiring harness and ease of installation, due to the fewer number of components to fit in. This concept is, however, limited by organisation, as it is not supported by recycling requirements [86]. Potential difficulties may be met because of the environment in which the ECU is required to work, for example under bonnet.

Another possible integration is between ECUs and their corresponding sensors and actuators [86, 90]. This would also lessen wiring cost.

The apparent advantage of peer-to-peer architecture over the conventional one is the reduced amount of wiring and hence cost. Historical information suggests the reduction in electronics costs, as opposed to the constant price of wiring harness. Therefore, the cost advantage of this architecture will even be more significant if the trend continues [93]. It is also based on a development of the existing autonomous systems, which it is logical to develop [89]. Now with the network capability built into some microcontrollers, the hardware becomes straightforward. Despite networking, each system is still independent. Reliability is high due to the fact that failure of one system does not affect other system functions. Moreover, having been networked, should a system failure happen, other ECUs would be able to detect and react to keep a vehicle running safely [94]. In terms of diagnostics, a specialised diagnostic ECU could merely be connected to the network [86,71]. By having centralised diagnostics, the separate task done by each ECU can be omitted, thus reducing the number of variations [86].

Data sharing and closed control corroboration between dynamical control ECUs result in an improved ride and handling, showing a potential benefits of the peer-to-peer architecture [28,96].

The considerations that need addressing with this architecture are cost and selection of appropriate network standards.

2.6.4 Master-Slave Networking

This architecture is similar in terms of hardware to the previous one, but has an extra centralised controller to oversee the operations of other local controllers. Hence the name master-slave. The diagram of this architecture is as shown in Figure 2.28 [86].

The function and scope of the master varies in level of control, and number of functions handled. At its most complex, the master controller could be connected by a bus to the entire vehicle's networks, including body control, powertrain, and telematics networks [86]. In the system shown in Figure 2.28, all other controllers remain fully responsible of their own tasks, whereas the master acts as a gateway to handle communication between different networks, and performs diagnostic tasks.

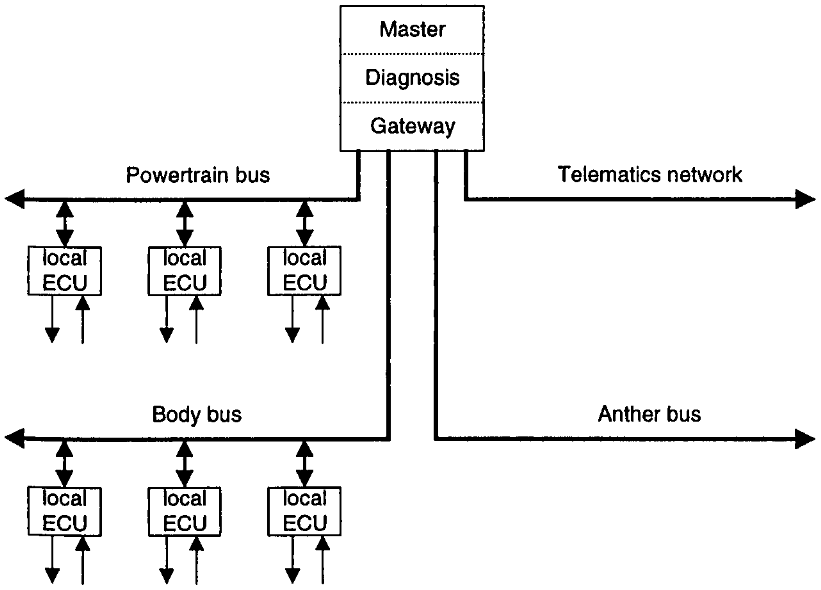


Figure 2.28 Master Slave network architecture

Emaus [90] describes a system with a master controller for each of the vehicle's networks. The engine area master controller receives information from local or slave engine, transmission and brake controllers, and issue system-level commands to them. The majority or all of the input processing, for example, switch debounce and data filtering, and output processing such as driving actuators, are the responsibility of local controllers. Hence the master slave architecture can be applied at a number of different levels.

Hettich [92] points out that this architecture is comparable to the hierarchical living model, where the master controller is equivalent to a cerebrum and various local controller units represent organs. The master controller is responsible for the strategic functions such as safety and diagnosis, where basic functions, for instance, braking and clutch controls are performed by local controllers. It is also suggested that considering just the local ECUs, development should be driven by seeking an optimum cost between the two extremes of networking local ECU with their mechanical units and integrating them into a combined unit.

This hierarchical concept is similar to that of Millward [89] who, however, only supports the idea of having individual local buses between peer ECUs.

Stevens [95] has implemented master-slave architecture on a test vehicle by having a master controller, diesel engine management, ABS, clutch management, and adaptive damper connected together by CAN bus. It was demonstrated that additional functions such as traction control, cruise control and vehicle stability control can be performed by the master controller with no extra ECUs needed and only small amount of additional hardware required.

The benefits of this master-slave arrangement are similar to those of the peer to peer networking. One additional benefit is in terms of cost saving and ease of introducing new functions with little or without any extra hardware. The amount of cost saving, however, depends on the complexity of a vehicle, as there is a trade off between the amount of new system cost saving and the cost of adding a master controller.

2.7 References

1. *Hartley J* **Automobile Steering & Suspension** Newnes Technical Books, 1977
2. *Newton K, et al.* **The Motor Vehicle 11th edition** Butterworths, 1989
3. *Gillespie T* **Fundamentals of Vehicle Dynamics** Society of Automotive Engineers, 1992
4. *von Glasner E, et al.* **Analysis of Intelligent Suspension Systems for Commercial Vehicles** SAE No.933008
5. *Milliken Jr. W* **Active Suspension** SAE No.880799
6. *Jurgen R* **Automotive Electronics Handbook** McGraw-Hill, 1995
7. *KojimaH, et al.* **Development of New Toyota Electronic Modulated Suspension - Two Concepts for Semi-Active Suspension Control** SAE No.911900
8. *Crolla D, et al.* **Semi-Active Suspension Control for a Full Vehicle Model** SAE No.911904
9. *Wallentowitz H* **Scope for the Integration of Powertrain and Chassis Control Systems: Traction Control - All-Wheel Drive - Active Suspension** SAE No.901168
10. *Hurdwell R, et al.* **Active Suspension and Rear Wheel Steering Make Powerful Research and Development Tools** SAE No.930266
11. *Iijima T, et al.* **Development of a Hydraulic Active Suspension** SAE No.931971
12. *Aoyama Y, et al.* **Development of the Full Active Suspension by Nissan** SAE No.901747
13. *Sugasawa F, et al.* **Electronically Controlled Shock Absorber System Used as a Road Sensor Which Utilises Super Sonic Waves** SAE No. 851652
14. *Tsutsumi Y., et al.* **Development of Pieze TEMS (Toyota Electronic Modulated Suspension)** SAE No. 901745
15. *Rover* **The New Range Rover Electric Manual**, Rover Group 1995
16. *Inagaki S, et al.* **Development of Feedforward Control algorithms for Active Suspension** SAE No. 920270

17. *Leighton N, et al.* **A Novel Active Suspension System for Automotive Application** Proceedings of the IMechE part D, Journal of Automotive Engineering V208 No. D4 1994
18. *Williams R* **Electronically Controlled Automotive Suspensions** Computing & Control Engineering Journal, June 1994
19. *Irie N, et al.* **4WS Technology and the Prospects for Improvement of Vehicle Dynamics** SAE No.901167
20. *Sato H, et al.* **Development of Four Wheel Steering System Using Yaw Rate Feedback Control** SAE No.911922
21. *Hirano Y, et al.* **Development of an Integrated System of 4WS and 4WD by H∞ Control** SAE No.930267
22. *Tanizaki S, et al.* **The Effect of Active Rear Steer System** International Symposium on Advanced Vehicle control (AVEC), 1998
23. *Kizu R, et al.* **Electronic Control of Car Chassis Present Status and Future Perspective** International Congress on Transportation Electronics, 1988
24. *Yokoya Y, et al.* **Integrated Control System Between Active Control Suspension and Four Wheel Steering for the 1989 CELICA** SAE No.901748
25. *Mitamura R, et al.* **System Integration for New Mobility** SAE No.881773
26. *Larsen R* **AVCS: An Overview of Current Applications and Technology** Proceeding of the Intelligent Vehicle Symposium, 1995
27. *Kageyama I, et al.* **An Advanced Vehicle Control Method Using Independent Four-Wheel-Steering System** IEEE Conference on Intelligent Transportation System 1997
28. *Kawakami H, et al.* **Development of Integrated System Between Active Control Suspension, Active 4WS, TRC and ABS** SAE No.920271
29. *Bischof H, et al.* **The ECU of A Rear Wheel Steering System** 8th International Conference on Automotive Electronics, 1991
30. *Adachi K, et al.* **Study on a New Four-Wheel-Steering Control Method At Low Speeds-Front-End Path Memorising Method** 8th International Conference on Automotive Electronics, 1991
31. *Eguchi T, et al.* **Development of "Super Hicas", a new rear wheel steering system with phasereversal control** SAE No.891978
32. *Oppenheimer P* **Comparing Stopping Capability of Cars with and without antilock Braking Systems (ABS)** SAE No.880324
33. *Lambourn R* **Braking and Cornering Effects with and without Anti-Lock Brakes** SAE No.940273
34. *Kolbe A, et al.* **Teves MK IV Anti-Lock and Traction Control System** SAE No.900208
35. *Limpert R* **Brake Design and Safety** SAE, 1992
36. *Chowanietz E* **Automotive Electronics** SAE, 1995
37. *Rittmannsberger N* **Antilock Braking System and Traction Control** International Congress on Transportation Electronics, 1998
38. *Mauer G, et al.* **Fuzzy Logic Continuous and Quantising Control of an ABS Braking System** SAE No.940830
39. *Voit M, et al.* **Methodology for the Design of a New Strategy in Vehicle Braking: Simulation and Comparison of Algorithms** Proceedings to the Advanced Vehicle Electronics Control (AVEC), 1994
40. *Jun C* **The Study of ABS Control System with Different Control Methods** Proceedings to the Advanced Vehicle Electronics Control (AVEC), 1998

41. *Schurr H, et al.* **A New Anti-Skid-Brake System for Disc and Drum Brakes** SAE No.840468
42. *Bauer H (chief editor)* **Automotive Brake Systems** Robert Bosch GmbH, 1995
43. *Denton T* **Automobile Electrical & Electronic Systems** Edward Arnold, 1995
44. *Dickerson W* **68HC05-Based System Design** Dr. Dobb's Journal, August 1995
45. *Masutomi S, et al.* **Development of ABS and Traction Control Computer** SAE No.901707
46. *McIntyre S* **Two Functions, One Microcontroller Four-Wheel ABS and Ride Control Using** 80C196KB SAE No.881138
47. *Wiehen C* **An Antilock Braking System for Sports Utility Vehicles and Light Trucks** SAE No.910698
48. *Hassain S* Digital **Algorithm Design for Wheel Lock control System** SAE No.860509
49. *Birch T* **Automotive Braking Systems 2nd edition** Saunders College Publishing, 1990
50. *Mathues T* **ABS Extending the Range** SAE No.940829
51. *Malone M* **The Microprocessor A Biography** TELOS, 1995
52. *Predko M* **Handbook of Microcontrollers** McGraw-Hill, 1998
53. *Hintz K, et al.* **Microcontrollers Architecture, Implementation, & Programming** McGraw-Hill, 1992
54. *Holdsworth B.* **Microprocessor Engineering** Butterworth, 1987
55. *Poole L.* **Inside the Microprocessor** MacWorld, October 1992
56. *Horowitz P, et al.* **The Art of Electronics 2nd edition** Cambridge University Press, 1990
57. *Bursky D* **High-Integration Controller Tackles Automotive and Industrial Needs** Electronic Design, April 6th 1998
58. *Weiss R* **16-bit Microcontrollers Take Over from 8-bitters** Computer Design, June 1996
59. *Jurgen R* **Automotive Microcontrollers** SAE, 1998
60. *Shah P, et al.* **Programmable Memory Trends in the Automotive Industry** SAE No.901133
61. *Kattwinkel T* **16-bit Microcontroller C167R with Full CAN Module Tuning Up Automotive Networks with the CAN Bus** Siemens Components Vol. 30, Iss. 6, 1995
62. *Caine J (Supervisor Advanced Powertrain Engineering, Ford)* **Private E-Mail** (JCaine1@Ford.Com) 2001
63. *Intel* **8XC196NT User's Manual** Intel June 1995
64. *Mitsubishi* **M16C/62 Microcontroller User's Manual** Mitsubishi, 1999
65. *Motorola* **MC68HC11 User's Manual** Motorola, 1995
66. *Motorola* **MPC555 User's Manual** Motorola 15th Sep 1999
67. *Bannatyne R* **MPC555 A New Benchmark in Real-time Embedded Control Solutions** Embedded System Engineering Feb/Mar 1999
68. *Burdock W, et al.* **EMC Design Check List** Rover Group, Oct 1994
69. *Intel* **Intel Pentium 4 Specifications** website http://www.intel.com/pentium4/, November 2000
70. *Bannatyne R* **Future Developments in Automotive Microcontrollers** (Published in Automotive Microcontrollers (PT-75)) SAE, 1998
71. *Kobayashi K, et al.* **Diagnostics for vehicle electronics present and future** Proceedings to the 1992 International Congress on Transportation Electronics (92C002), 1992

72. *SAE* **Vehicle Network for Multiplexing and Data Communications Committee Glossary of Vehicle Networks for Multiplexing and Data Communications** SAE J1213 Part 1
73. *France R* **New Standards Help Vehicle Control Units Communicate** Electronic Product Design, November 1997
74. *Navet N* **Controller Area Network CANs Use Within Automobiles** IEEE Potential, October/November 1998
75. *Anonymous* **Controller Area Network** Hitex (UK) Ltd., 1996
76. *Bosch* **CAN Specification Version 2.0** Robert Bosch GmbH, 1991
77. *Bauer H (chief editor)* **Automotive Handbook 4th Edition** Robert Bosch GmbH, 1996
78. *Nath N* **CAN Protocol Eases Automotive-Electronics Networking** EDN Vol. 43 Issue 17, 1998
79. *Embacher M* **A Cost-Effective CAN MCU - Solution** Electronic Product Design, Vol. 17, Issue 4 1996
80. *Szydlowski C* **Tradeoffs Between Stand-alone and Integrated CAN Peripherals** SAE No.941655
81. *Kiencke U, et al.* **Open Systems and Interfaces for Distributed Electronics in Cars (OSEK)** SAE No.950291
82. *Motorola* **Motorola CSIC Microcontrollers** Motorola, Quarter 3, 1997
83. *Intel* **Embedded Microcontrollers** Intel, 1998
84. *Ohr S* **Safety and security spearhead changes in automotive electronics** Computer Design, January 1996
85. *Kiencke U, et al.* **Architectural trends in automotive electronics** IFAC advances in automotive control, 1995
86. *Holzinger O, et al.* **Automotive electronics - integration and partitioning** (92C028) Proceedings to the 1992 international congress on transportation electronics, 1992
87. *Fenton J* **Focus on networking of on-board vehicle electronic systems** Automotive Engineer, June/July 1996
88. *McLaughlin R, et al.* **A feasibility study of CAN technology in body electronic control systems** (C498/35/028) I MECH E Autotech, 1995
89. *Millward J* **Vehicle electronic system architectures - influences and guidelines** SAE No. 930010
90. *Emaus B* **Aspects and issues of multiple vehicle networks** SAE No. 950293
91. *Hansson H, et al.* **BASEMENT: a distributed real-time architecture for vehicle applications** Real-time systems Vol. 11 Issue 3, November 1996
92. *Hettich G* **A new approach for electronic-controller-functions in vehicles** SAE No. 960624
93. *Schubotz H* **Modular body systems** (C524/101/97) I MECH E Autotech, 1997
94. *Nakamura S et al.* **The high-speed in-vehicle network of integrated control system for vehicle dynamics** SAE No. 910463
95. *Stevens S* **The development and testing of an integrated systems demonstrator vehicle** (C498/35/157/95) I MECH E, 1995
96. *Sherman D* **Electronic Chassis Control** Automotive Industry, August 1999
97. *Appleyard M, et al.* **Active Suspension: Some Background** IEE Proceedings – Control Theory Applications Vol.142 No.2, March 1995
98. *Yong S, et al.* **Feed Forward Neuro-Controlled Active Suspension Using Frequency and Time-Mixed Shape Performance Index** International Journal of Vehicle Design Vol. 17 No. 2, 1996

99. *Soliman A, et al.* **Preview Control for a Semi-Active Suspension System** International Journal of Vehicle Design Vol. 17 No. 4, 1996
100. *Esmailzadeh E, et al.* **Optimal Active Vehicle Suspensions with Full State Feedback Control** SAE No.922473
101. *Tanaka H, et al.* **Development of a Vehicle Integrated Control System** ImechE C389/220, 1992
102. *Hoogterp F et al.* **An Energy Efficient Electromagnetic Active Suspension System** SAE No.970385
103. *Intel* **83C196EA Microcontroller User's Manual** Intel, May 1996
104. *Motorola* **RISC Central Processing Unit Reference Manual Revision 1** Motorola, February 1999
105. *Motorola* **Microcontroller Selector Guide** Motorola Quarter 1, 2001
106. *Di Vincenzo F, et al.* **Global Trends Towards Systems** 30th International Symposium on Automotive Technology & Automation Vol. 2, 1997
107. *Melbert J* **Next Generation Automotive Electronics: The Impact on Technologies and Design Methodologies** 1992 Symposium on VLSI Circuits Digest of Technical Papers, 1992
108. *Kopetz H* **Automotive Electronics** Proceedings of the 11th EUROMICRO Conference on Real Time Systems, 1999
109. *Jurgen R* **The Electronic Motorist** IEEE Spectrum, March 1995
110. *Ranta R, et al.* **The Importance of Electronics in Modern Cars** Proceedings to IEMC '96 Managing Virtual Enterprises: A Convergence of Communications, Computing, and Energy Technologies, 1996
111. *Cambou B* **Microsystems for Automotive Application - Semiconductor Aspects** Microsystem Technologies Vol. 3 Issue 3 Springer-Verlag 1997
112. *Tindell K* **Embedded Systems in The Automotive Industry** Embedded Systems Conference, 1999
113. *Lawrence P, et al.* **Real-Time Microcomputer System Design: An Introduction** McGraw-Hill, 1987
114. *Motorola* **Automotive Selector Guide** Motorola, Quarter 4, 2000

CHAPTER 3

DERIVATION OF DIFFERENT AUTOMOTIVE ELECTRONIC CONTROL SYSTEMS & TEST DATA

As stated in the introduction, this chapter describes the exemplary dynamic control systems that are used to represent ABS, suspension control and 4WS. In the next chapter, these systems will then be arranged into a vehicle model according to different automotive electronic architectures.

3.1 Exemplary Systems

ABS, suspension control and 4WS have been chosen as the modelled dynamic control systems. ABS has gained a significant acceptance in the market in the past decade. A quarter of new cars world-wide are reported to be fitted with ABS [1]. ABS is also expected to exist commonly in future medium and small sized vehicles, as it is in the luxury and family car market at present. With suspension control and 4WS, the acceptance is still far below that of ABS. Electronic suspension controls are still mostly installed in only luxury vehicles, while 4WS are most popular among Japanese car manufacturers. However, with both systems' obvious benefit to ride and handling of a vehicle, with additional advantages shown when used as integrated control, they are expected to achieve greater production scale, if cheaper components and manufacturing techniques become available.

These dynamic control systems are to be used as exemplary systems for modelling the alternative vehicle architectures in the next chapter. These systems are based upon the ones which are available in the market as they are proved practical. Also advanced systems are needed, as they can closely represent the systems of the near future, in terms of electronic contents and complexity. Specific information demanded for this modelling is the method of control and electronic implementation, which have been taken from published descriptions of their operation

The three exemplary systems are first described in terms of their overall functionality and component contents, followed by a more detailed examination of their control algorithms.

3.1.1 Exemplary ABS System

The ABS system selected for modelling in this thesis is based on the ABS and traction control system described in [2]. The system adopts the same control strategy as mainstream ABS systems, which is deceleration threshold. The system is chosen because it is an actual system fitted in a vehicle in the market with the available information on system signals and wiring provided in the literature. Although the details of its control algorithm are not explicitly discussed, as is in all the literature in this field, it indicates that the deceleration threshold method is employed. Since the aim of the deceleration threshold control method is to maintain wheel speed or wheel slip, or both, within the target values, all the ABS controls of this method follow similar steps. The same type of control algorithm described in details in [14] is, therefore, used as an ABS algorithm control model. The details of this ABS control will be described in the functioning section of this chapter.

This exemplary ABS system is electronically controlled by a centralised ECU and employs a hydraulic system as its actuator. The simplified electrical block diagram of this ABS system is shown in Figure 3.1.

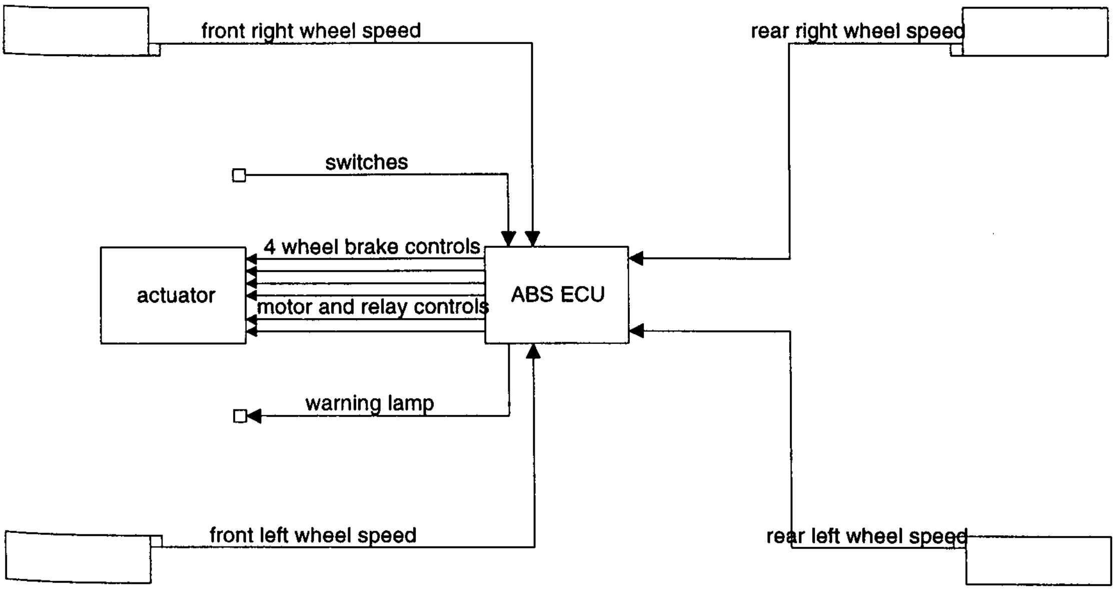


Figure 3.1 Simplified electrical diagram of an exemplary ABS system

3.1.1.1 Exemplary ABS Electronic Control

Wheel speed signals are sine waves of varying frequencies, proportional to the wheel speeds. The range of frequencies of the wheel speed signals is approximately 18-2000 Hz [2]. Signal and timing characteristics of the system require that a wheel speed and acceleration are calculated every 6 ms [2]. The same timing is required for ABS control calculation. The ABS control model is described later.

3.1.2 Exemplary Suspension Control System

The exemplary suspension system chosen as a model in this project is one of the few fully active suspensions currently available in the market. This is because of higher power requirements, more complexity, greater expense and higher weight of active suspension compared to the variety of more popular semi-active suspension systems. However, theory suggests that active suspension can achieve the highest performance among all the suspension systems [7]. Furthermore, it is believed that future technology will enable electric active suspension, which would ease system complexity and weight penalty problems of current hydraulic active suspension systems. In terms of electronics, active suspension, due to its complexity, requires high processing power for control. The system is, therefore, considered suitable as a model for an electronically complex and demanding control systems of the future.

3.1.2.1 Exemplary Active Suspension Electronic Control

The active suspension system is described in [3,4]. A hydraulic system is used as an actuator in this system. The system is based around a centralised ECU for system control. Six accelerometers, one longitudinal, two lateral and three vertical, sensing all three dimensional movements of the vehicle, together with four vehicle height sensors and a vehicle speed sensor, each situated near each wheel, provide data for the control. Control signals are designated to a multi-valve unit and hydraulic pressure source of pump and motor, which maintain the overall system pressure. Ride height of each wheel is controlled by another control signal from the ECU to a wheel pressure control valve. The control cycle is not specifically stated, but is said to be 'several milliseconds' [3]. This active suspension control algorithm and electrical block diagram of this active suspension system is displayed in Figure 3.2 [3,4].

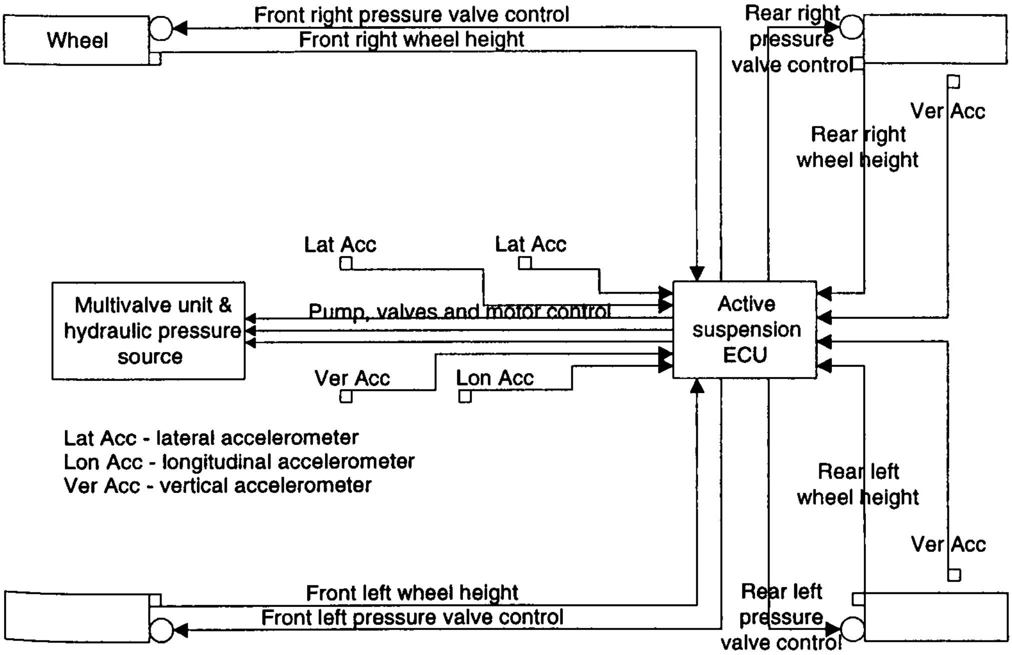

Wheel
Front right pressure valve control
Front right wheel height
Rear right pressure valve control
Ver Acc
Rear right wheel height
Lat Acc
Lat Acc
Multivalve unit & hydraulic pressure source
Pump, valves and motor control
Active suspension ECU
Ver Acc
Lon Acc
Lat Acc - lateral accelerometer
Lon Acc - longitudinal accelerometer
Ver Acc - vertical accelerometer
Rear left wheel height
Ver Acc
Front left wheel height
Front left pressure valve control
Rear left pressure valve control

Figure 3.2 Simplified electrical diagram of an exemplary suspension control system

3.1.3 Exemplary 4WS System

An exemplary 4WS system used for modelling in this project is one of the most advanced 4WS systems fitted in production cars. It is described in [5,6]. In the previous chapter, classifying 4WS systems according to their methods of actuation, later systems are of electronic-electric control and electronic-hydraulic-mechanical systems. Despite their differences, these advanced 4WS systems share similar closed loop control algorithms, using proportional control with vehicle speed and steering angle as major input data. Additional input data is steering speed or yaw rate for improved control. Due to this similarity, their electronic control systems can be assumed to have the same level of complexity. The system selected is chosen because it has more details of its electronic control and components available in literature than other advanced 4WS systems of the same complexity [5-10]. This information is essential for modelling purpose.

3.1.3.1 Exemplary 4WS Electronic Control

The exemplary 4WS system has a centralised ECU, which receives data from vehicle speed sensor, yaw rate sensor and front wheel steering angle sensor for determining a rear steer angle. At low speeds, the rear wheels are steered up to five degrees in the opposite direction to the front wheels, to reduce turning radius. At medium to high speeds, the rear wheels are steered to a relatively small angle to the same direction as the front wheels, to improve stability. The rear wheel steer control signal is sent to a rear steering actuator. Closed loop proportional control is applied in this 4WS system. The control details are explained in the previous chapter and also in the functioning section of this chapter. The simplified electrical block diagram of this 4WS system is displayed in Figure 3.3 [5,6].

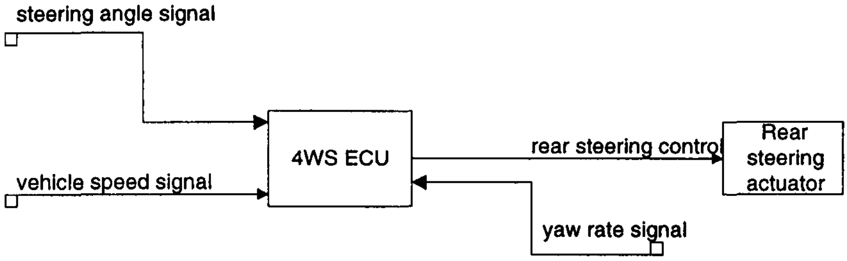


Figure 3.3 Simplified electrical diagram of an exemplary 4WS control system

3.2 Exemplary Dynamic Control System Functionality

In order to verify the operation of these exemplary ABS, active suspension, and 4WS systems, a simulation of each of the systems is made using MATLAB *Simulink*. The control algorithm of each system is first interpreted as a *Simulink* model. Road test data obtained from a vehicle making a circuit of the Gaydon Emissions Circuit was used to drive the inputs of the systems. Some data, such as acceleration data, steering angle, etc. was not available and hence has been synthesised from the available information. The resulting system output signals, representing control commands to actuators, are then inspected to see how the systems respond to those sensor data. Since the system functional characteristics are broadly known (such as ABS would activate when a wheel shows a sign of locking, or active suspension would maintain stable vehicle ride height), the output signals can verify these characteristics and the correct operation of the systems.

3.2.1 ABS Control Model

The nature of ABS control is very complex. A large number of different rules are set by system developers, to cater for various braking circumstances a vehicle would encounter. These rules are used to activate the ABS only to prevent an imminent wheel lock, without unnecessary ABS operation which would sacrifice valuable braking distance. The difference in braking characteristics between driven and non-driven wheels with and without gear engaged, and the change in vehicle speed relative to wheel speed at medium and heavy braking, to name a few, also contribute to the greater number of rules for ABS control. Furthermore, because vital data for ABS calculation, such as vehicle speed and wheel slip cannot be measured directly, representative figures have to be obtained from wheel speed data. Different ABS manufacturers develop their own set of rules and the methods of these representative value calculations. This is all undisclosed information. The ABS control described in this thesis is hence a simplified model formed from general ABS control descriptions available in literature.

The ABS model is adapted from deceleration threshold control, which is the most common ABS control algorithm [11]. The specific ABS control model used follows the control method is described in [14]. The control procedures are as shown in Figure 3.4 [14]. The figure represents ABS control on a high traction surface, for example dry asphalt, which is slightly different from the control on low traction surface, such as icy road. The ABS will compare each wheel acceleration with the deceleration threshold, -a. The ABS will not sacrifice the valuable braking distance, by activating to reduce brake pressure when the wheel acceleration first goes below the deceleration threshold. It will only reduce brake pressure when both wheel acceleration and wheel slip are lower than the threshold of peripheral wheel deceleration, -a, and slip switching threshold, λ_1, respectively. Brake pressure will be decreased until wheel acceleration goes back above the deceleration threshold, as seen in phase 3.

v_F Vehicle speed, v_{Ref} Reference speed, v_R Peripheral wheel speed, λ_1 Slip switching threshold, $+A$,$+a$ Thresholds of peripheral wheel acceleration, $-a$ Threshold of peripheral wheel deceleration, $-\Delta p_{ab}$ Brake-pressure decrease.

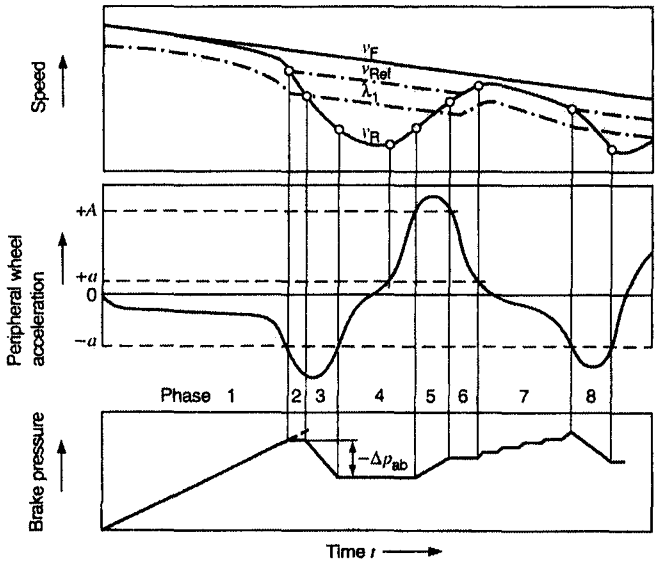


Figure 3.4 ABS control cycle

Since the way in which the ECU determines road surface condition is unknown, and the fact that the control concepts for different traction surface control are the same, i.e. deceleration threshold, the ABS model will be based on the high traction surface control cycle. A *Simulink* model of the general ABS system was devised by the author and is as shown in Figure 3.5

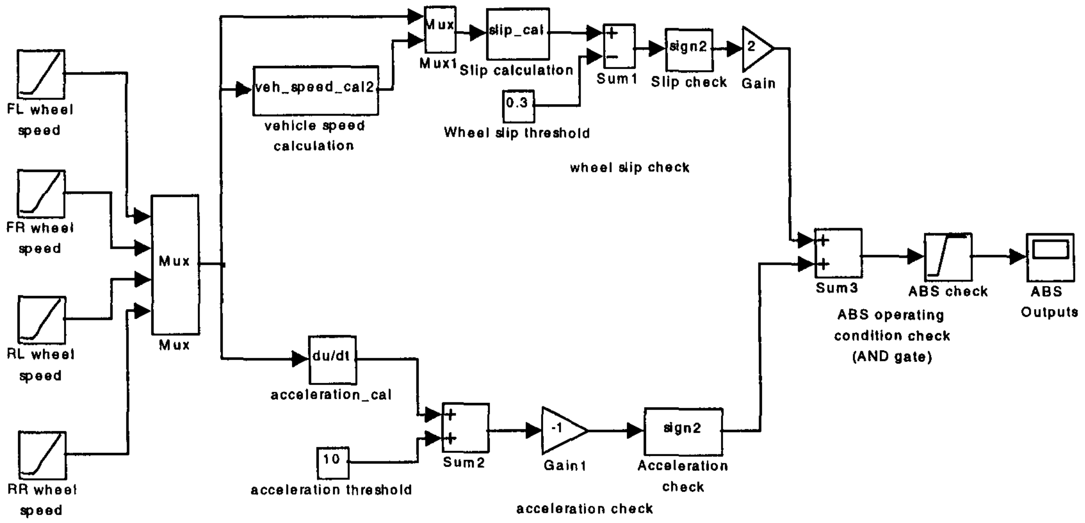


Figure 3.5 *Simulink* model of the ABS system

In the model, wheel speed signals already converted from sine waves are fed into the model. In order to avoid a repetition of calculation, the four wheel speed signals are multiplexed throughout the whole model. Reference vehicle speed is calculated from an average speed of a pair of diagonal wheels. From it, wheel slips are also determined, based on the equation 3.1 below, which is described in the previous chapter.

$$\lambda = (V_v - V_r) / V_v \qquad (3.1)$$

λ - wheel slip
V_v - vehicle velocity
V_r - circumferential wheel speed

The equation can be applied in all cases except when the vehicle is stationary i.e. vehicle speed is zero. That case is beyond the operation condition of the ABS. Wheel decelerations or accelerations are also computed. Wheel slip and acceleration are then compared to the threshold values, in wheel slip and acceleration check sections, respectively. When both wheel slip and acceleration drop below the thresholds, as monitored by the AND gate equivalent, the ABS issues output command to reduce brake pressure. Low values are selected as thresholds for demonstration purpose.

3.2.2 Active Suspension Control Model

The aim of this active suspension control is to exert active power to counteract longitudinal, lateral forces and also braking and driving torques by the tyres. It also acts to minimise vehicle body roll. In short, the control system tries to keep the four wheels level at appropriate height in all driving situations.

The exemplary suspension control system employs a proportional control method as shown in Figure 3.6. The K_{xx} elements are the control gains and the term $\frac{1}{1+TS}$ are a first order delay for vertical acceleration. A judgement circuit is there to determine if ride height is within an allowable range. The vertical, longitudinal and lateral accelerations multiplied with their respective gains and delays, and wheel heights are added. The sum of the values is the output for each wheel height control.

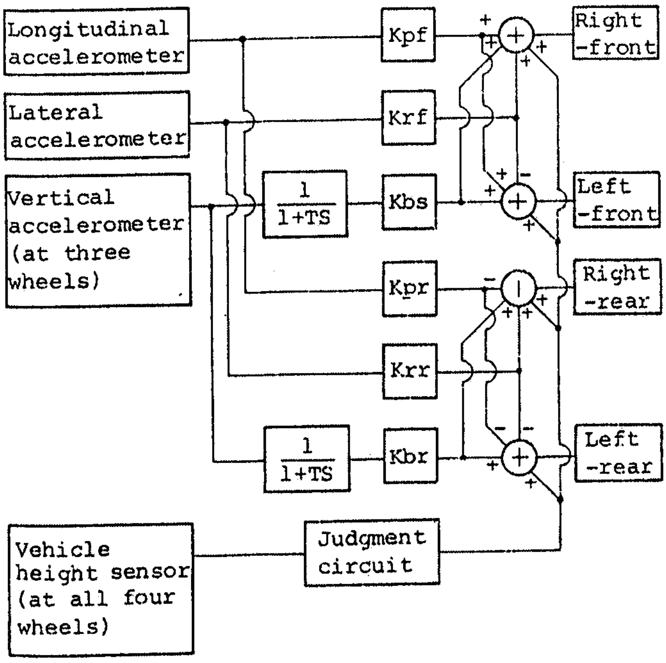


Figure 3.6 Exemplary active suspension control block diagram

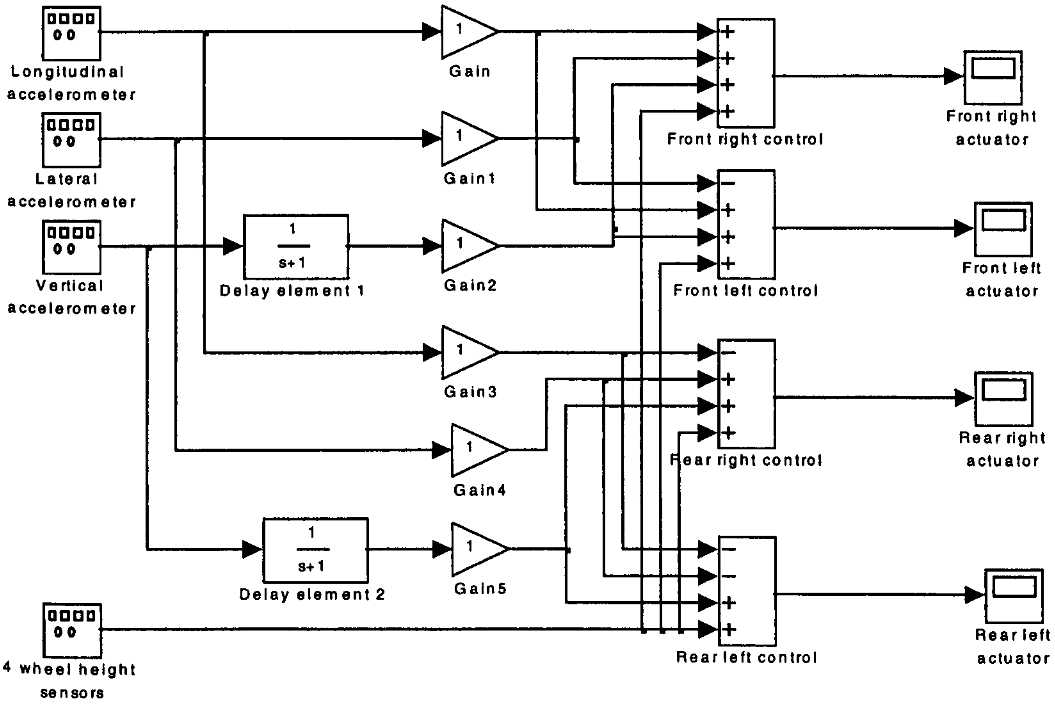


Figure 3.7 *Simulink* model of active suspension system

This has been modelled by the author, as a *Simulink* model of the active suspension control system, and is displayed in Figure 3.7. The judgement circuit logic which performs a diagnostic function is not available. Its description is to check if the wheel heights are within range. It can, therefore, be omitted by assuming wheel heights are within allowable range, which is what is expected under normal road driving. The values of the control gains and delay elements are not known. The purpose of this simulation is to demonstrate the functioning of the control systems only, and no quantitative results are required. All the control gains and delay elements are therefore set to one.

3.2.3 4WS Control Model

The concept of a 4WS system operation is to steer the rear wheels in an opposite direction to the front wheels at low speeds, while the rear wheels are turned in the same direction as the front wheels at medium to high speeds. The opposite steer is to reduce turning radius, whereas the same directional steer is to improve handling and stability.

In the exemplary 4WS system, a more recent control method is modelled, such that instead of controlling the rear wheels according to the driver steering inputs alone i.e. steering angle and speed, they can additionally take account of the current state of the vehicle [6]. By doing so, the control system would be more robust to changing driving conditions, such as crosswind and varying road conditions. The vehicle current state can be indicated by its yaw rate. Hence a yaw rate sensor is introduced to sense vehicle deflection due to sudden braking, side wind, or road surface irregularity at medium to high speeds. Rear wheels are steered according to yaw information to maintain vehicle zero slip angle, and hence minimise the effect of the disturbances and improve stability. At low speed however, the rear wheels are steered to angles opposite to the front wheels' angles to improve vehicle manoeuvrability.

Proportional control is employed in the system. A control map with steering angle and yaw rate proportional terms, are used to determined the 4WS system control. Three control gains: opposite-direction steering angle proportional, steering velocity tuning and yaw rate proportional gains are present. The second gain has the value according to steering and vehicle speeds, whereas the other two have their values corresponding to vehicle speed. Equation 3.2 is the control governing formula [6]. The control block diagram is shown in Figure 3.8 [5].

$$\theta_r = K_1(V).\theta_f + K_2(\dot{\theta}_f, V).K_3(V).\omega_y \qquad (3.2)$$

$\dot{\theta}_f$: front wheel steering angle velocity

θr : rear wheel steering angle

θf : front wheel steering angle

V : vehicle speed

ω_y : yaw rate

$K_1(V)$: opposite direction steering angle proportional gain

$K_2(\dot{\theta}_f, V)$: tuning gain of steering velocity

$K_3(V)$: yaw rate proportional gain

From the control block diagram, the steering velocity tuning and yaw rate proportional gains are set to zero at low speed, making the second term of the equation zero. Hence at low speed driving, 4WS is control by the first term, steering angle proportional term, with the opposite direction steering angle proportional gain K_1. The negative K_1 indicates that the rear wheels are steered in the opposite direction to the front wheels. At medium to high speed, K_1 becomes zero, leaving the control to the second term, the yaw rate proportional term. The rear wheel steer direction depends on vehicle yaw, as the system acts to reduce the yaw rate [6]. Hence the effect of road disturbances is minimised.

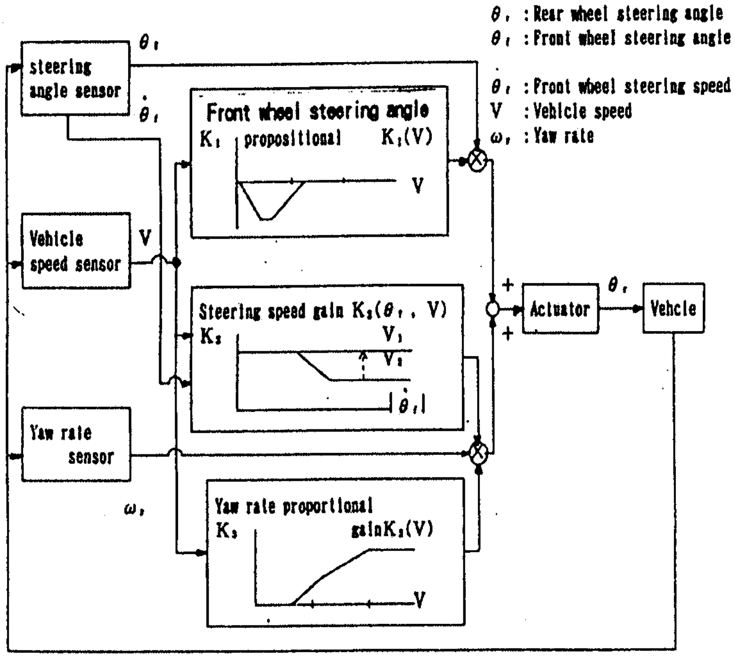


Figure 3.8 Exemplary 4WS system control block diagram

A *Simulink* model of the above system has been devised by the author and is shown in Figure 3.9.

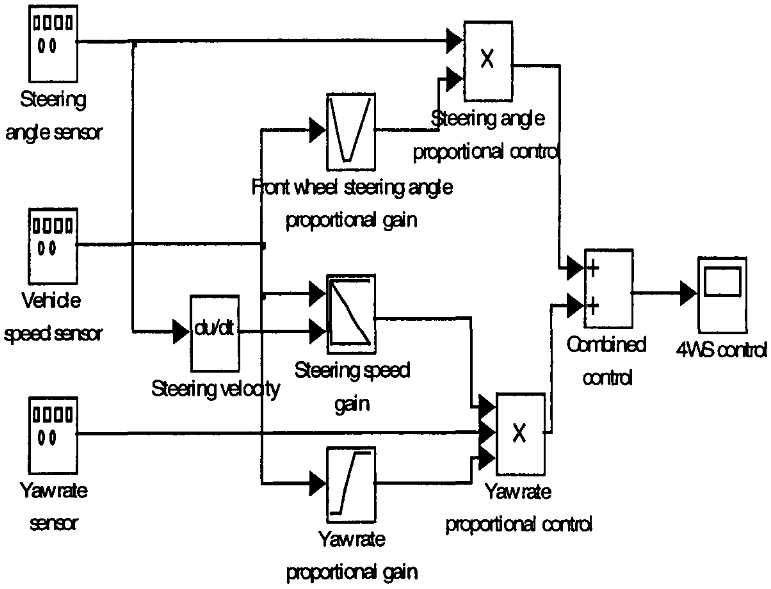

0000
00
Steering angle sensor
Vehicle speed sensor
Yaw rate sensor
du/dt
Steering velocity
X
Steering angle proportional control
Front wheel steering angle proportional gain
Steering speed gain
Yaw rate proportional gain
Yaw rate proportional control
+
+
Combined control
4WS control

Figure 3.9 *Simulink* model of 4WS system

This *Simulink* model is an exact representation of the block diagram in Figure 3.8. The gain values are not known, so they are arbitrarily set for the simulation, with the gain shapes maintained.

In order to exercise the systems described above, a set of road test data was used.

3.3 Road Test Data

The input data for the above control system simulation is produced and generated based on a road test data. The data was collected by R. J. Ball on 6th Feb 1992, from driving a Rover Metro around the Rover Emissions test track at Gaydon, England. The map of the track is as shown in Figure 3.10. The data was collected at approximately every 0.1s, over the period of 4 minutes. The types of data collected were time, engine speed, sync manifold pressure, throttle angle and road speed, of which only time and road speed are of use for the simulation. The rest of the input data such as steering angle, vertical and lateral accelerations, have had to be generated based on the available data and the track map.

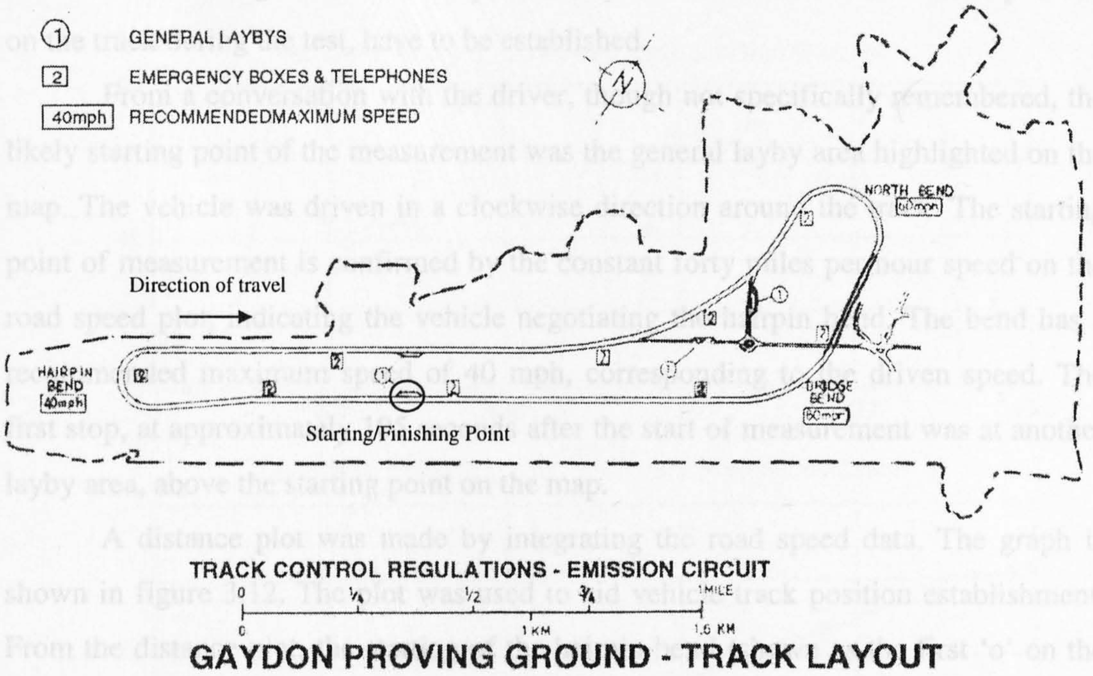

① GENERAL LAYBYS
2 EMERGENCY BOXES & TELEPHONES
40mph RECOMMENDEDMAXIMUM SPEED
N
NORTH BEND
60mph
Direction of travel
HAIRPIN BEND
40mph
BRIDGE BEND
60mph
Starting/Finishing Point
TRACK CONTROL REGULATIONS - EMISSION CIRCUIT
0
1/2
3/4
1MILE
1 KM
1.5 KM
GAYDON PROVING GROUND - TRACK LAYOUT

Figure 3.10 Map of Rover's Emissions test track at Gaydon

3.3.1 Data Preparation

After a small number of spurious errors, which are single zero road speed samples during high speed driving (believed to be an instrumentation fault) are removed, the road speed data is produced as shown in Figure 3.11.

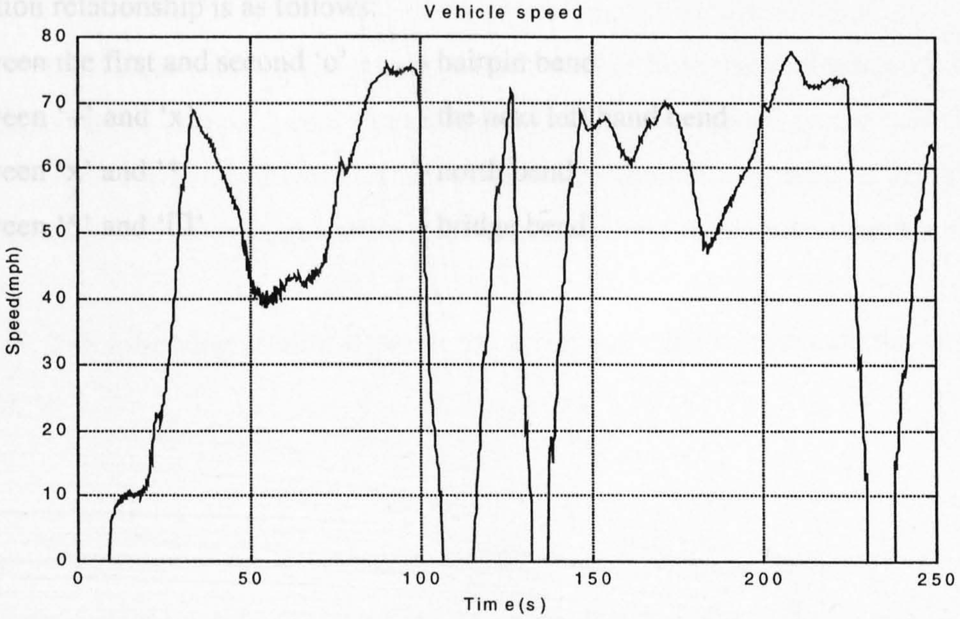


Figure 3.11 Vehicle speed data produced from road test

In order to generate other input data, specific information of vehicle position on the track during the test, have to be established.

From a conversation with the driver, though not specifically remembered, the likely starting point of the measurement was the general layby area highlighted on the map. The vehicle was driven in a clockwise direction around the track. The starting point of measurement is confirmed by the constant forty miles per hour speed on the road speed plot, indicating the vehicle negotiating the hairpin bend. The bend has a recommended maximum speed of 40 mph, corresponding to the driven speed. The first stop, at approximately 105 seconds after the start of measurement was at another layby area, above the starting point on the map.

A distance plot was made by integrating the road speed data. The graph is shown in figure 3.12. The plot was used to aid vehicle track position establishment. From the distance plot, the starting of the hairpin bend (shown as the first ‘o’ on the plot) is specified as half a mile from the starting point, as measured from the track layout. The second ‘o’ on the plot indicates the end of the hairpin bend. This is again, established by measuring the length of the bend from the track map. The next left hand bend is measured to be 0.8 mile from the hairpin bend. The process is then repeated to identified all vehicle track positions. The vehicle track positions in curves are marked with different symbols shown in Figure 3.12. The symbols and vehicle position relationship is as follows:

between the first and second ‘o’	- hairpin bend
between ‘+’ and ‘x’	- the next left hand bend
between ‘x’ and ‘*’	- north bend
between ‘*’ and ‘□’	- bridge bend

1

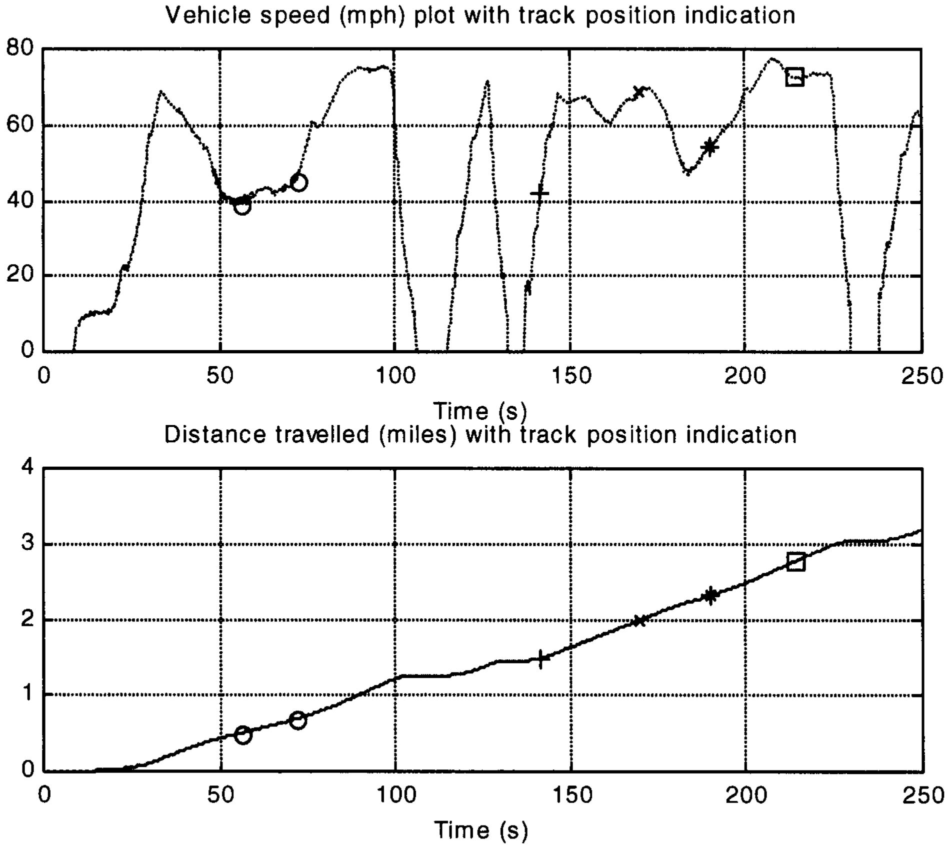


Figure 3.12 Vehicle speed and distance travelled with track position indication

3.3.2 Wheel Speed Data

Individual wheel speed data was not available from the road test, so it has been generated from vehicle speed. The four wheel speeds are based on the vehicle speed with a slight variations created by an added noise. The noise represents the difference in speed sensor readings, due to the causes analysed in the fishbone diagram of Figure 3.13.

The following section explains the procedure used to calculate the size of this noise.

3.3.2.1 Wheel Speed Noise

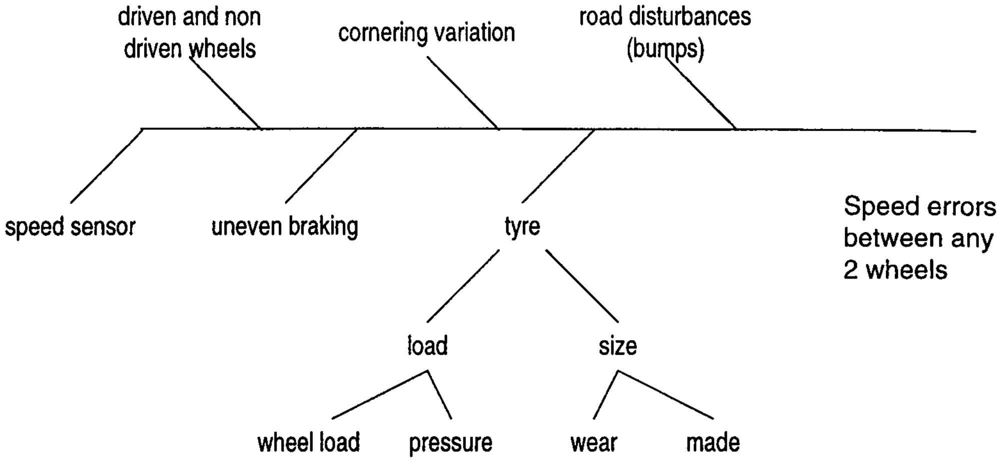

driven and non driven wheels
cornering variation
road disturbances (bumps)
speed sensor
uneven braking
tyre
Speed errors between any 2 wheels
load
size
wheel load
pressure
wear
made

Figure 3.13 Fishbone diagram of the causes of wheel speed difference

The causes and their effects to the wheel speed difference are:

- speed sensor. Speed sensors fitted to vehicles are of variable reluctance or hall effect, and recently introduced magnetic resistance element or optical types [7]. They all produce signals of magnitude or frequencies proportional to their rotational speed. With the straight relationship between them, the error in measurements is insignificant compared to other sources.
- The driven wheels are subjected to drive torque from the engine. This extra force, in effect, flattens the tyres more, hence reducing their rolling circumference. The driven wheels, therefore, roll faster than the non-driven ones during acceleration. When braking with low gear engaged, the driven wheels are subjected to engine torque, causing increase in wheel mass moment of inertia. This makes the driven wheels react slower to changes of braking torque in an unstable braking region [14]. Since there is no excessive braking, which would cause an ABS system to operate, in the test drive, all the braking is confined within a stable region. Hence there is assumed no difference between driven and non-driven wheels during braking.
- road disturbances (bumps). Bumps on the road may cause a wheel to be temporarily off the ground. The effect is a slow down of wheel speed. Since the test track is smooth, the effects of the disturbances are expected to be small.
- cornering variation between outer and inner wheels. The effect is that the outer wheels spin faster than the inner ones during bends. This effect is included in the following section.

- uneven braking. Uneven braking could cause a particular wheel to decelerate faster or slower than other wheels, depending on heavier or lighter brake grip and the road surface. Since the test track is smooth and the test vehicle is in good condition, the effect of an uneven braking is assumed small.
- tyre
  - wheel load. A wheel with heavier load would be flatter to the ground. This makes it effective rolling circumference smaller, hence turning faster. A wheel can be subjected to heavier or lighter load than others. Hence the effect could be either faster or slower speed.
  - tyre pressure. An underinflated tyre has the same effect as a heavy loaded wheel i.e. having smaller rolling circumference, and hence turning faster than normally inflated tyres. An overinflated tyre has the opposite effect. Since normal drivers tend to inflate their tyres to a specified pressure and reinflate when the pressure drops, the tyres are underinflated most of the time. This results in higher measured wheel speeds, but the effect is likely to be common to all four tyres.
  - tyre wear. A reduction in tyre tread depth means smaller tyre circumference. The test vehicle was fitted with tyres of specification 155/65 R13. Its radius with no load is (13 x 2.54 x 10) / 2 + 155 x 0.65 = 265.85 mm. An approximate maximum allowable tyre tread wear of 10mm would reduce the wheel radius by (10 / 265.85) x 100 = 3.76%. Hence this is also the same percentage of possible variation in wheel speed.
  - tyre make. Tyres of the same specification made by different manufacturers will be the same nominal size, but will have minor differences in rolling radius. The test vehicle was a low-mileage vehicle owned by the manufacturer and had the same brand of tyre fitted on each wheel.

3.3.2.1.1 Effects of road disturbances and uneven braking

As mentioned earlier, the variation in wheel speed measurements due to road disturbances and uneven braking are small, due to the quality of the test track and vehicle. The effect is, therefore, assumed negligible in the simulation.

3.3.2.1.2 Effects of different tyre conditions

The wheel speed measurement variation, caused by different tyre circumstances discussed above, is considered almost constant during a single trip. This is because the changes in tyre wear, tyre pressure and load can be ignored during a four minute drive. The effect is assumed small and constant, as considered previously in a similar work in the literature [15]. A random number from a normal distribution of mean 1 and variance 0.01 (equivalent to 1% of the wheel speed), is used as an multiplying offset to a series of the speed measured from each wheel.

3.3.2.1.3 Effects of an engine on the driven wheels

As previously discussed, the driven wheel would roll faster than the non-driven wheel during acceleration. This is due to wheel slip, which is required to generate a tractive force between tyre rubber and road surface. Similar to braking, the amount of tractive force available depends on the tyre-road coefficient of friction. Consider the slip against friction (tractive) force plot in Figure 3.14, friction force increases with the greater slip up to the maximum at approximately 15-20% slip on dry surface according to [12], and 10-15% according to [16]. Beyond this point, braking or acceleration will enter an unstable region, which if continues, would result in wheel locking or spinning, respectively.

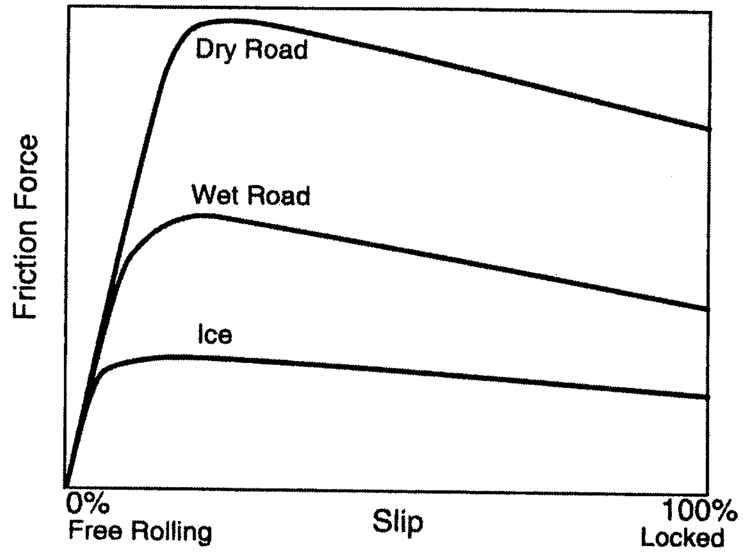


Figure 3.14 Friction force against wheel slip graph

During test drive, there was no extreme manoeuvring, hence the vehicle can be considered to operate in a stable range at all time. It is, therefore, assumed that the driven wheel speed data is subject to up to 15% slip.

Figure 3.15 shows a plot of normalised longitudinal force and wheel slip of different tyre loads. The longitudinal force and the slip ratio are normalised by dividing them with the frictional force. Hence the graph can be viewed as a longitudinal force against wheel slip plot.

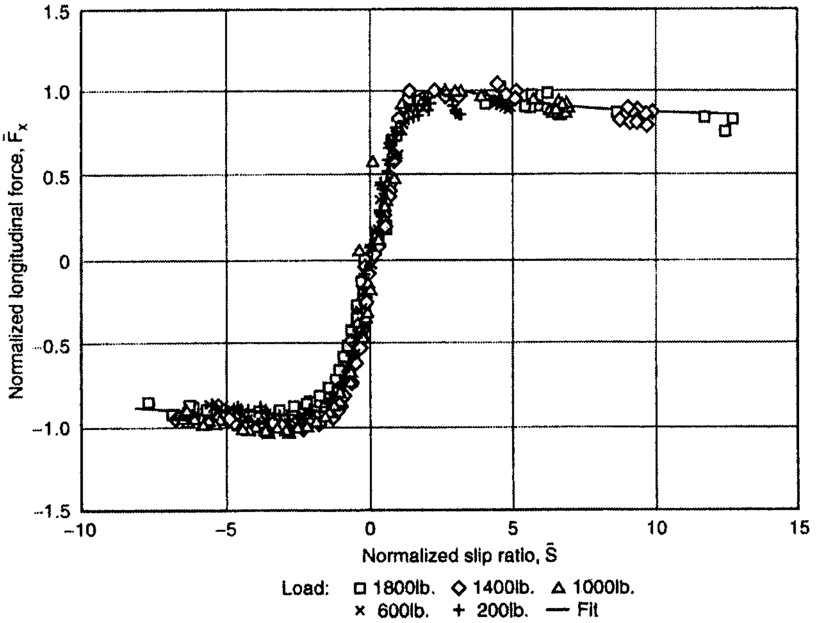


Figure 3.15 Normalised longitudinal force against normalised wheel slip plot

As discussed above, it is assumed that the vehicle tyres are within stable range throughout the drive. From the plots in Figure 3.14 and 3.15, the longitudinal force and slip are linear in the stable region. During low to medium speed, all the resistance forces on the vehicle are omitted, to calculate an acceleration using Newton's Second Law [12].

$$F = Ma$$

F : longitudinal force
M : vehicle mass
a : acceleration

A longitudinal force is linear with a wheel slip. An acceleration is, therefore, linear with a wheel slip. At low to medium speed, the relationship between acceleration and wheel slip is hence expressed as:

$$\lambda = c.a \qquad (3.3)$$

λ : wheel slip
a : acceleration
c : constant

Assuming that 15% slip (spin) occurs at the acceleration of 10 m/s^2, c is equal to 0.015. The plot of Figure 3.15 is based on the SAE wheel slip definition, which is opposite in sign to equation 3.1. By substituting equation 3.3 into the wheel slip equation, the following equation is obtained:

$$V_r = V_v\,(0.015a + 1) \qquad (3.4)$$

Hence the front wheel speed data during acceleration at the speed below 40 mph, is modified from the vehicle speed data using equation 3.4. Acceleration at high speed driving is subject to significant air resistance force, and other factors such as tyre pressure. Complex empirical model is needed to approximate the relationship [12]. Furthermore, during acceleration higher magnitude of wheel slip occurs at low speed, due to the higher driving torque of low gears. Therefore, at higher speed during acceleration, the front wheel speed data is modified to have a random slip (spin) of below 5%.

3.3.2.1.4 Effects of cornering

During cornering the outer wheel speeds are greater than the inner wheel speeds. The speed difference is calculated from the difference between distance travelled. For example, during the North Bend cornering:

North Bend is approximated to have half circle shape.

North Bend radius measured from the map	= 0.0867 miles
Assuming wheel track	= 1.6 metres = 0.001 miles
Radius of outer wheel travel (r)	= 0.0867 + 0.0005 = 0.0872 miles
Outer wheel travelling distance	= πr = 0.27395 miles
Inner wheel travelling distance	= 0.27081 miles
Average travelling distance	= vehicle travelling distance = (outer wheel travel - inner wheel travel) ÷ 2 = 0.27238 miles

Difference between outer wheel and vehicle travels $= \frac{(outerwheeltravel - vehicletravel)}{vehicletravel} x100\%$

$= 0.5764\%$

Difference between inner wheel and vehicle travels = 0.5764%

Half of the amount of difference is added to the vehicle speed data during in North Bend to form the outer wheel speed data. Likewise, half of the difference subtracted from the vehicle speed data represents inner wheel speed data. The same calculation process was also carried out for other bends.

By including the difference in speed between inner and outer wheels during cornering, and adding small randomly generated noise mentioned earlier, the four wheel speed data is created as shown in figure 3.16.

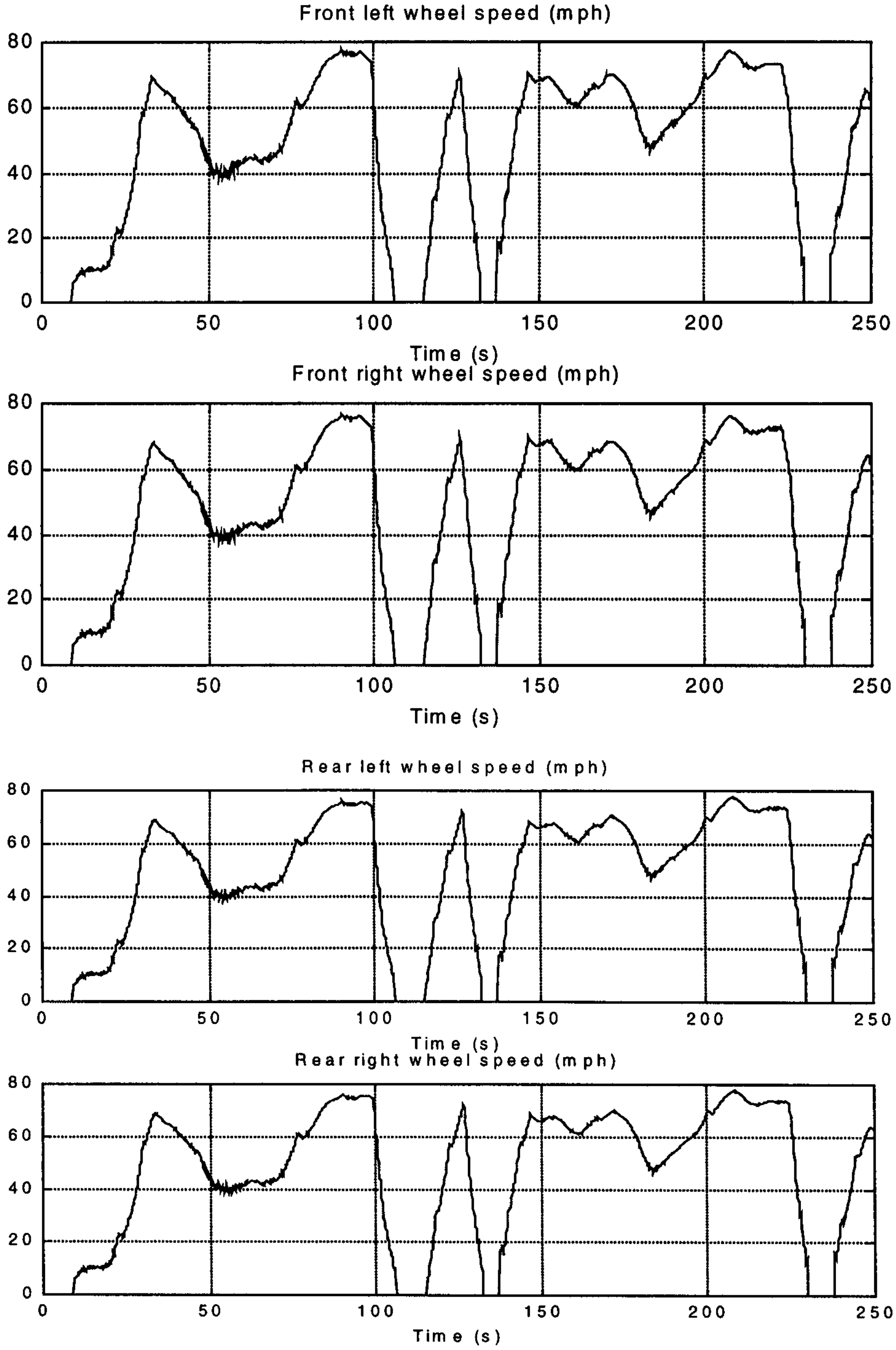


Figure 3.16 Four wheel speed data

3.3.3 Wheel Height Data

Since no information is available for wheel height during the road test, a wheel height is randomly generated during the drive and set to zero during stops. Throughout the data, a small amount of noise is added. Figure 3.17 shows the wheel height data, together with the vertical acceleration data generated from it.

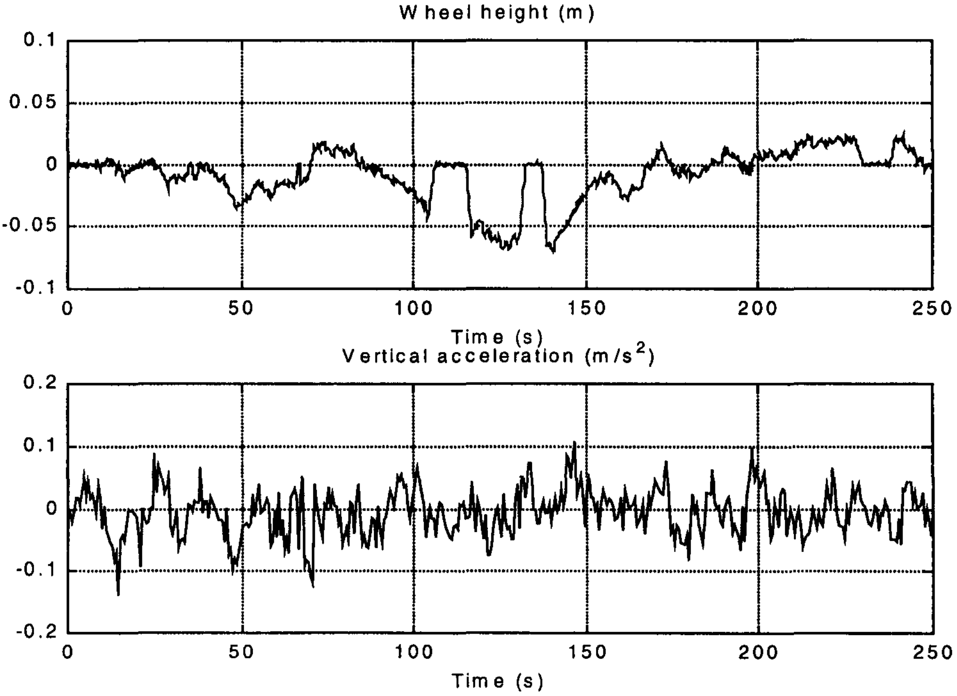


Figure 3.17 Wheel height and vertical acceleration data

3.3.4 Vertical Acceleration Data

Vertical acceleration data is generated from the wheel height data. Instead of taking only the second order derivative of the wheel height data, an average of the current and three other sample delays is taken, for the vertical acceleration. This, in effect, smoothes the vertical acceleration data. The vertical acceleration data is generated using the model shown in Figure 3.18. The vertical acceleration plot is shown in Figure 3.17.

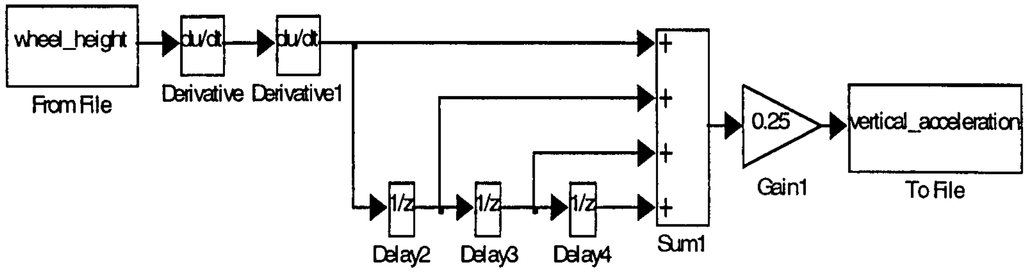


Figure 3.18 Model used to generate vertical acceleration data from the wheel height data

3.3.5 Longitudinal Acceleration Data

Longitudinal acceleration data is created by differentiating the vehicle speed data. Before the differentiation, the vehicle speed is converted into metre/second unit.

Figure 3.19 shows the longitudinal acceleration of the test vehicle.

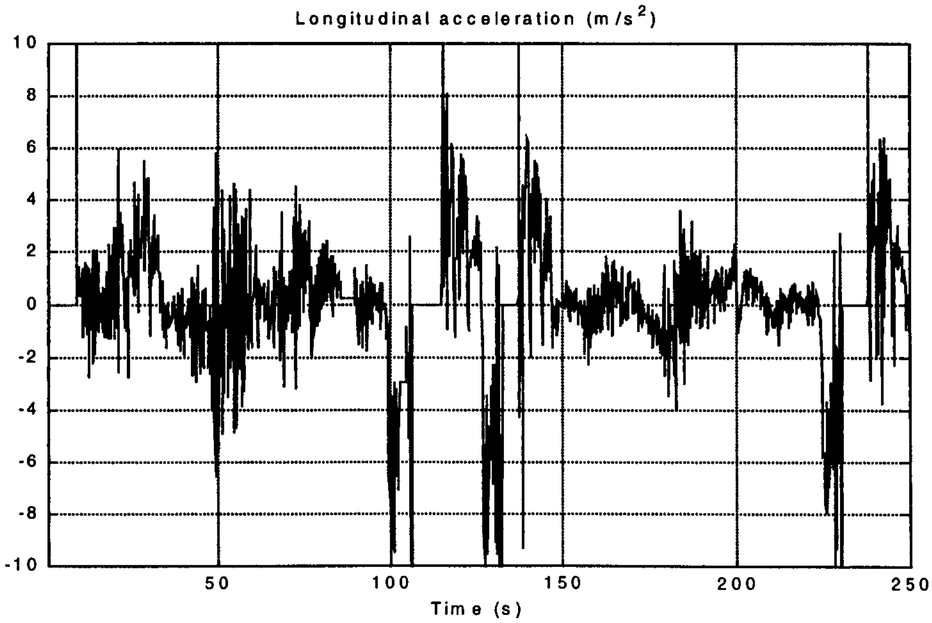


Figure 3.19 Longitudinal acceleration data

3.3.6 Lateral Acceleration Data

A vehicle experiences lateral acceleration when being steered from straight ahead travelling direction. From the road test, larger lateral accelerations would occur during cornering. The lateral acceleration of the road test is calculated. All the bends are assumed parts of circles. Equation 3.5 governs the movement of a particle travelling in a circle.

$$F = ma = m\frac{v^2}{r} \qquad \text{Hence } a = \frac{v^2}{r} \qquad (3.5)$$

F : centrifugal force
m : particle mass
a : lateral acceleration
v : particle speed
r : radius of the curve

The radii of the track bends are known. The lateral acceleration of the vehicle during bends is, therefore, calculated using the equation 3.5. On straight driving, lateral acceleration is calculated from the equation 3.6 [12].

$$a = \frac{\delta V^2 g}{57.3Lg + KV^2} \qquad (3.6)$$

δ : front steering angle (deg)
L : wheel base (ft)
a : acceleration (m/s^2)
K : understeer gradient (deg/g)
V :vehicle speed (m/s)
g :gravitational constant ≅ 9.81 (Nm/s)

The lateral acceleration is plotted in Figure 3.20.

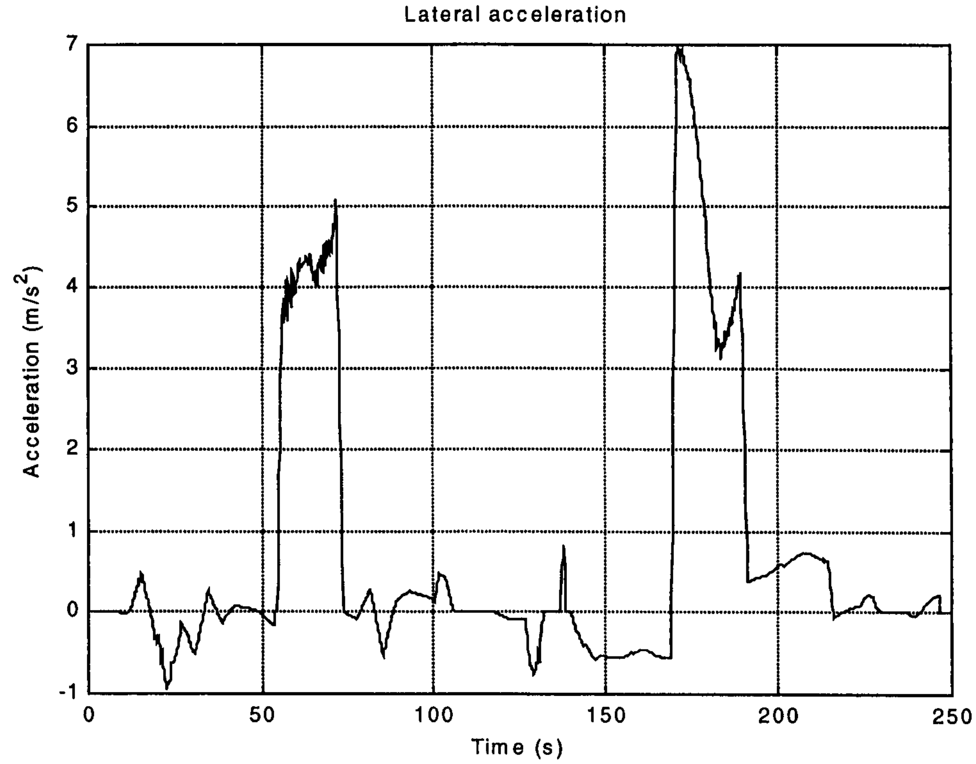


Figure 3.20 Lateral acceleration data

3.3.7 Steering Wheel Angle Data

The steering wheel angle data is independently generated for the section of the drive in bends and on other parts of the track.

During cornering in bends, the steering angles can be found from the equation 3.7 governing the steering angle during high speed cornering [12].

$$\delta = 57.3\ L/R + Ka_y \qquad (3.7)$$

δ : front steering angle (deg)
L : wheel base (ft)
R : radius of turn (ft)
K : understeer gradient (deg/g)
a_y : lateral acceleration (g)

From the study by Riede P, et al. [16], statistical data shows an approximately linear relationship between understeer gradient and vehicle curb weight. The test vehicle has a curb weight of 840 kg. Its approximate understeer gradient, obtained from the curb weight against understeer gradient plot, is 2.25 deg/g.

The vehicle wheel base is 2.27 metres [14]. An example of the front steering angle calculation for during a 60mph drive in the North Bend is as follows:

Radius of bend (R) = 0.0867 miles
= 0.0867 x 1600 = 138.72 metres

From equation 3.7, L and R have to have the same unit, so R is converted into metres.

Wheel base (L) = 2.27 metres

From equation 3.6, lateral acceleration (a) = $V^2 / R = 60^2 \times (1600/3600)^2 / 138.72$
$= 5.126\ m/s^2 = 5.126 / 9.81 = 0.523\ g$

Front steering angle (δ) = $(57.3 \times 2.27/138.72) + 2.25 \times 0.523$
= 2.114 degrees

Using equation 3.7, the front wheel steering angle of the vehicle is calculated for rides in all the bends.

On other parts of the track, the front steering angle is randomly generated to be within the scope of the graph obtained from [13]. Lechner, et al. [13] studied the steering characteristics in relation to vehicle speed, based on measurements from actual road driving. It shows that as the vehicle speed is higher, the less steering is made. The relationship is shown in Figure 3.21. It should be noted that in the actual test track drive, there is much less steering than in actual driving, because there are no sharp turning points as normal streets, nor overtaking manoeuvres. However, the aforementioned steering characteristic study is used to generate the steering data for non-bending part of the track, in order to create an exaggerated steering angle data. The exaggerated steering angle data can demonstrate the 4WS operation at low speed drive more clearly than a more realistic, almost zero degree steering data.

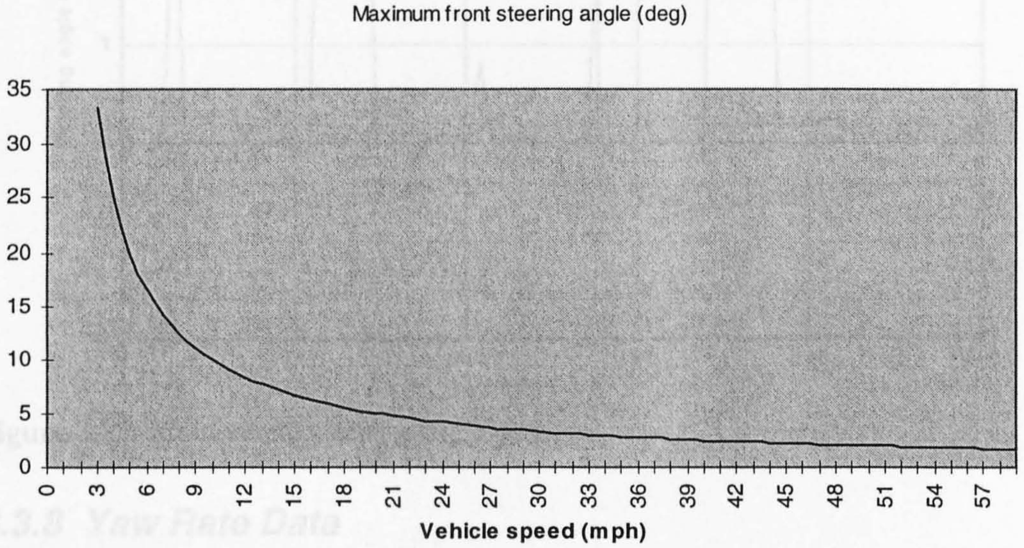


Figure 3.21 Relationship between maximum front steering angle and vehicle speed

Based on this plot, the steering angle is randomly generated within the envelope of the plot and the x axis.

Combining the front wheel steering angle generated during cornering and the rest of the drive together, the complete front wheel steering data is plotted in Figure 3.22.

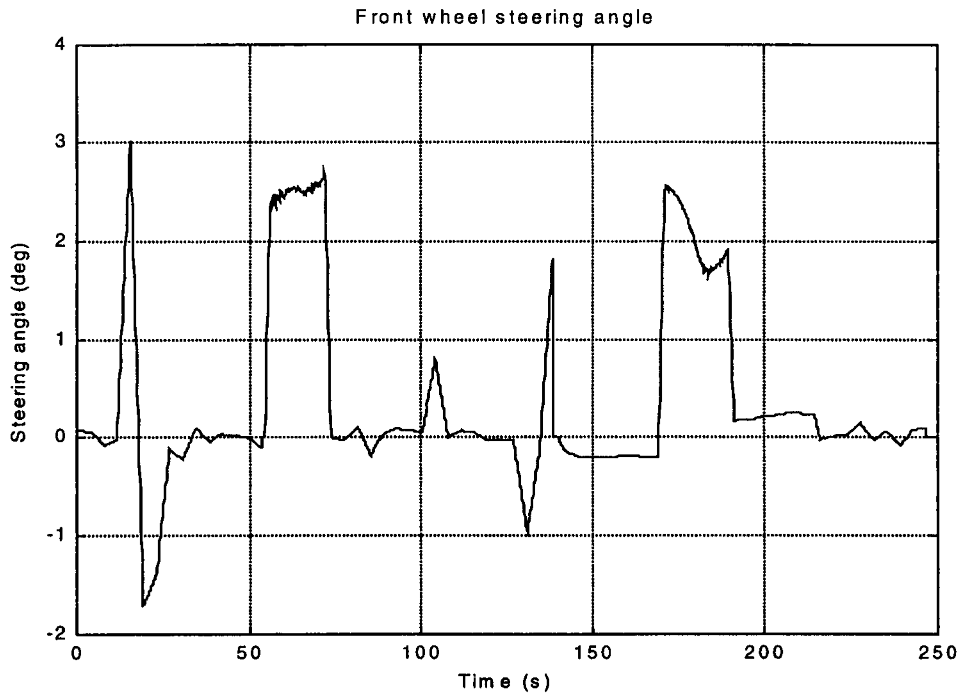


Figure 3.22 Front wheel steering angle data

3.3.8 Yaw Rate Data

The yaw rate during cornering can be found using the equation 3.8 [12]

$$r = \frac{57.3 g \delta V}{57.3 L g + K V^2} \tag{3.8}$$

r : yaw rate (deg/s)
V : vehicle speed (m/s)
δ : front steering angle (deg)
L : wheel base (ft)
K : understeer gradient (deg/g)
g :gravitational constant ≅ 9.81 (Nm/s)

Yaw rate calculation at speed 50mph during the North bend cornering is as follows:

Radius of turn (bend)	= 0.0867 miles
Vehicle speed	= 50 / 3600 (miles/s)

The distance unit of V and R have to be the same.

Yaw rate	= 57.3 x (50/3600) / 0.0867
	= 9.18 deg/s

Outside the bends, yaw rate is randomly generated. Figure 3.23 displays the yaw rate plot.

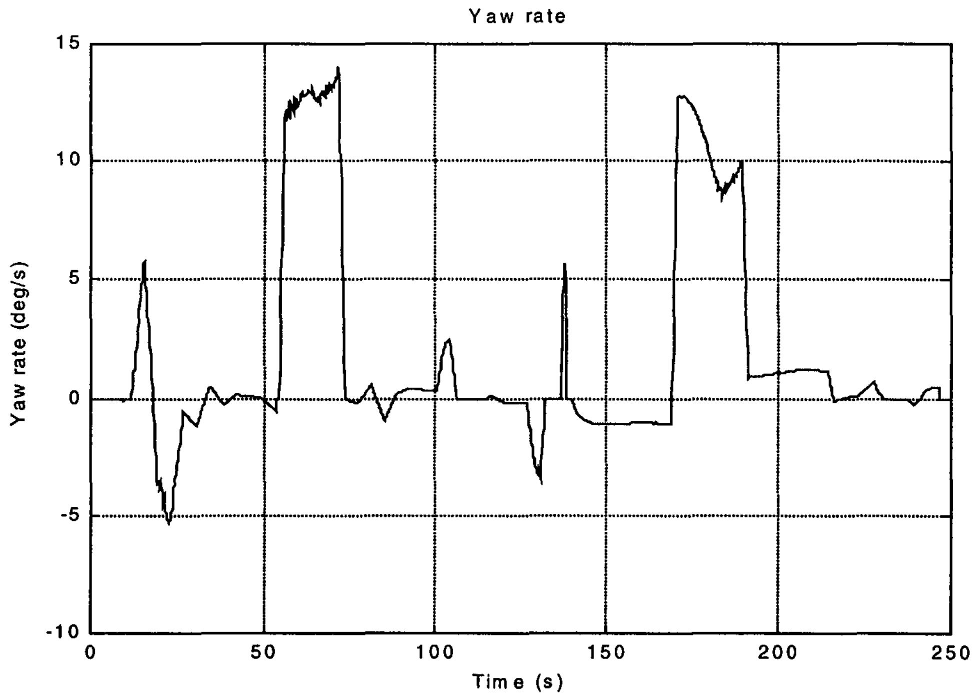


Figure 3.23 Yaw rate data

3.3.9 Whole Vehicle Model

Now all the data required is available. It will be fed into the whole vehicle dynamic model, and the system response will be used to verify the control system operations. The whole vehicle model of the dynamic control systems is shown in Figure 3.24.

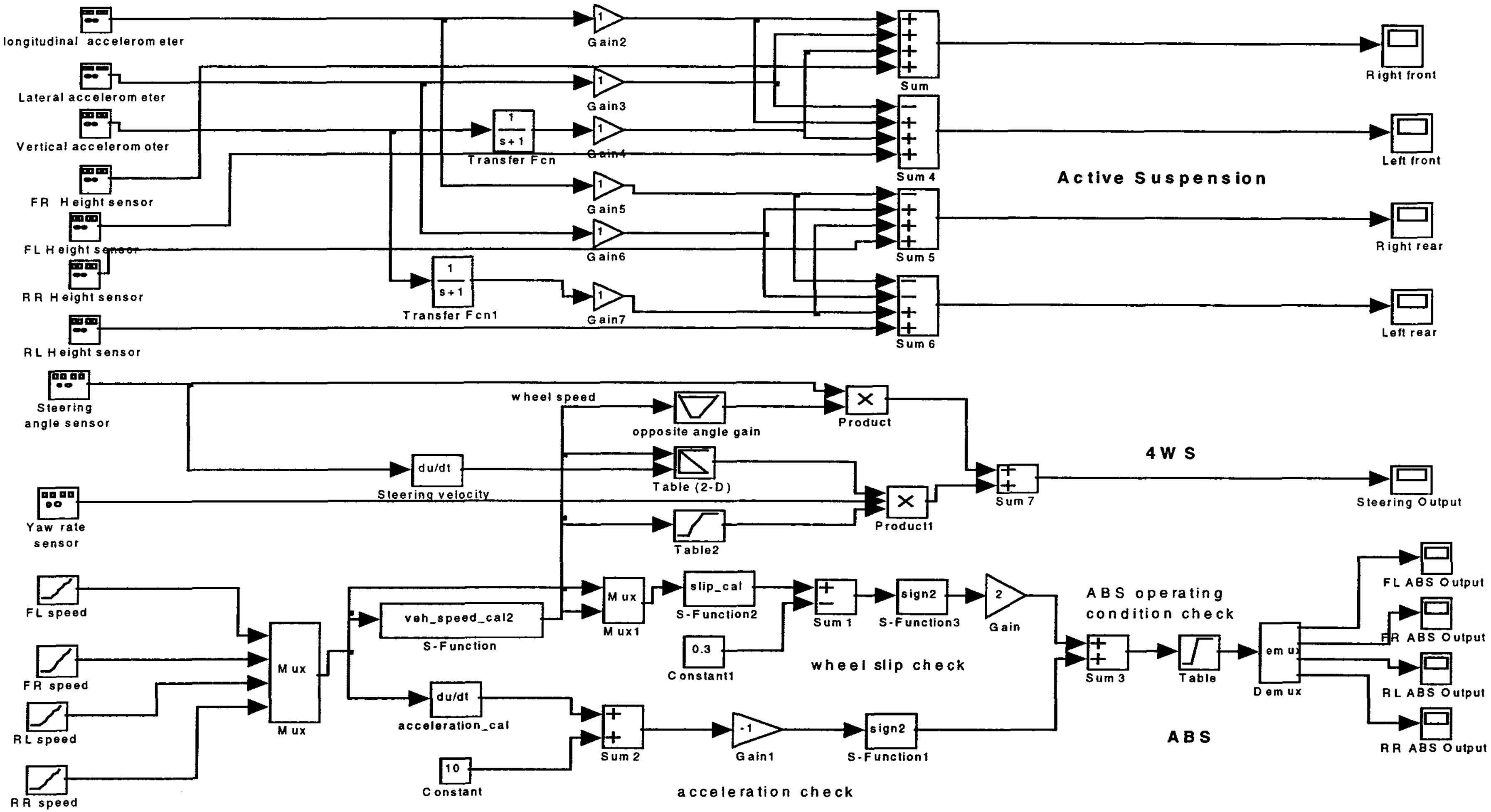


Figure 3.24 *Simulink* vehicle model of the dynamic control systems

3.4 Whole Vehicle Simulation

All the sensor signals produced were input into the whole vehicle model and the system responses were obtained. For clarity of system verification, ABS, active suspension and 4WS operations will be analysed separately.

3.4.1 ABS Simulation

The inputs to the ABS are the four wheel speed sensors and the response is the ABS commands for the four wheel control. The plot of the inputs and the responses, from the simulation, are shown in Figure 3.25 and 3.26, respectively.

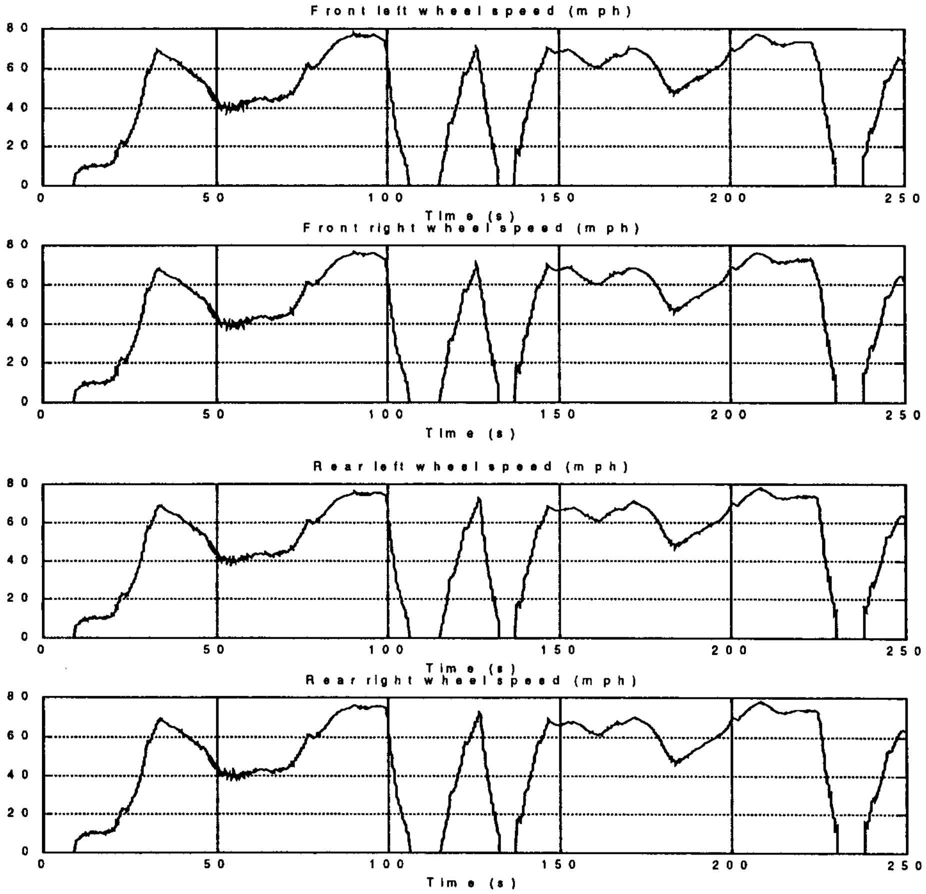


Figure 3.25 Four wheel speed sensor inputs

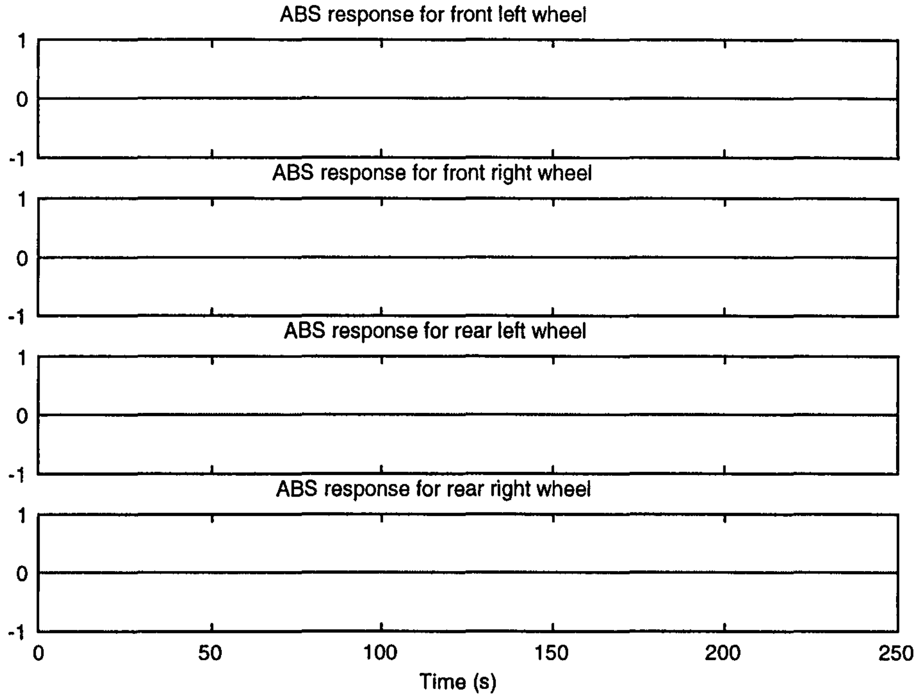


Figure 3.26 ABS responses

In this case, none of the wheels show signs of locking, so the ABS does not activate.

This does not allow ABS operation to be demonstrated during the test data, hence to demonstrate ABS operation, the front left wheel speed data is modified to have higher deceleration rate. Its plot together with the front right wheel speed data is shown in Figure 3.27. The part modified is the deceleration around 100th second. The ABS responses are plotted in Figure 3.28.

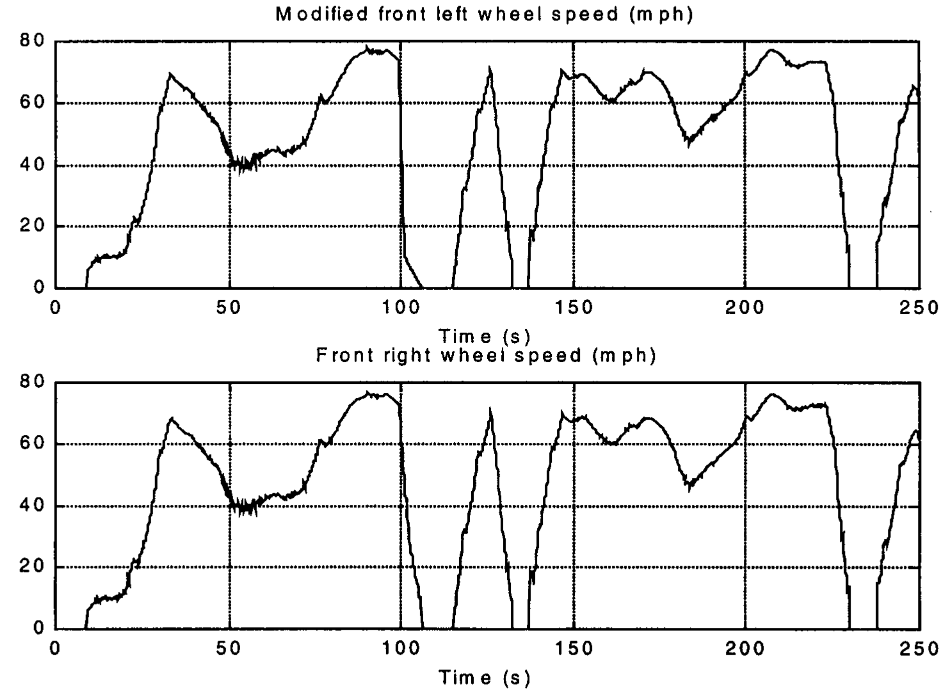


Figure 3.27 Modified front left wheel speed to show sign of wheel locking

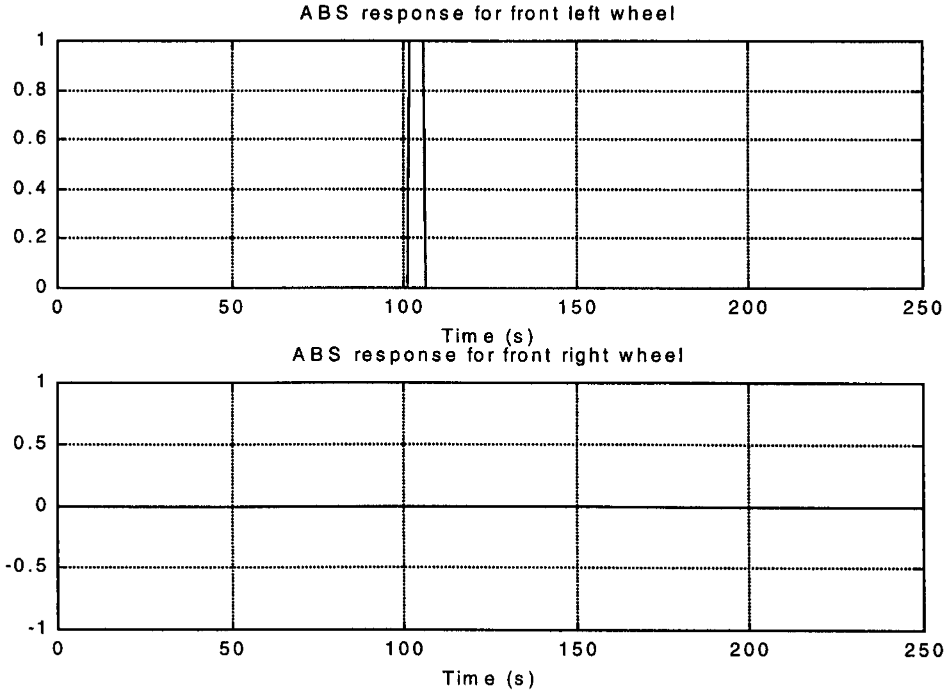


Figure 3.28 ABS responses to front left wheel locking

The response plot of the front left wheel demonstrates that when the ABS detects an impeding wheel lock, it reacts by sending a command to that wheel actuator to prevent it. During normal braking of the front right wheel, the ABS detects no sign of locking, and hence leaves the front right wheel actuator inactivated.

3.4.2 Active Suspension Simulation

For the active suspension system, the three dimensional accelerations and the wheel height signals are the inputs. Its responses are the four wheel height demand signals. The inputs and responses are plotted in Figure 3.29 and 3.30, respectively.

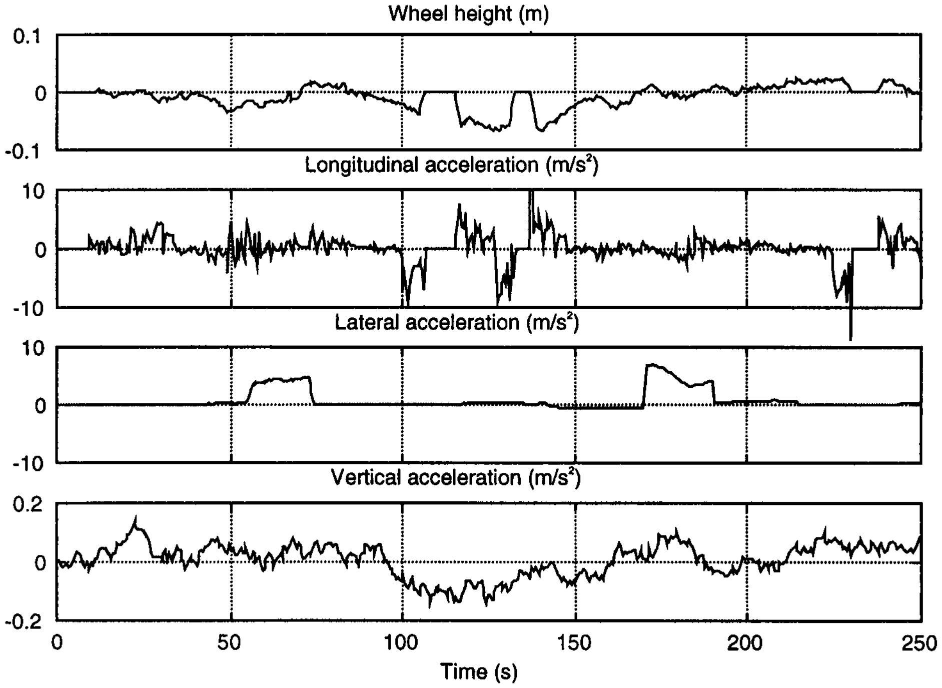


Figure 3.29 Active suspension inputs

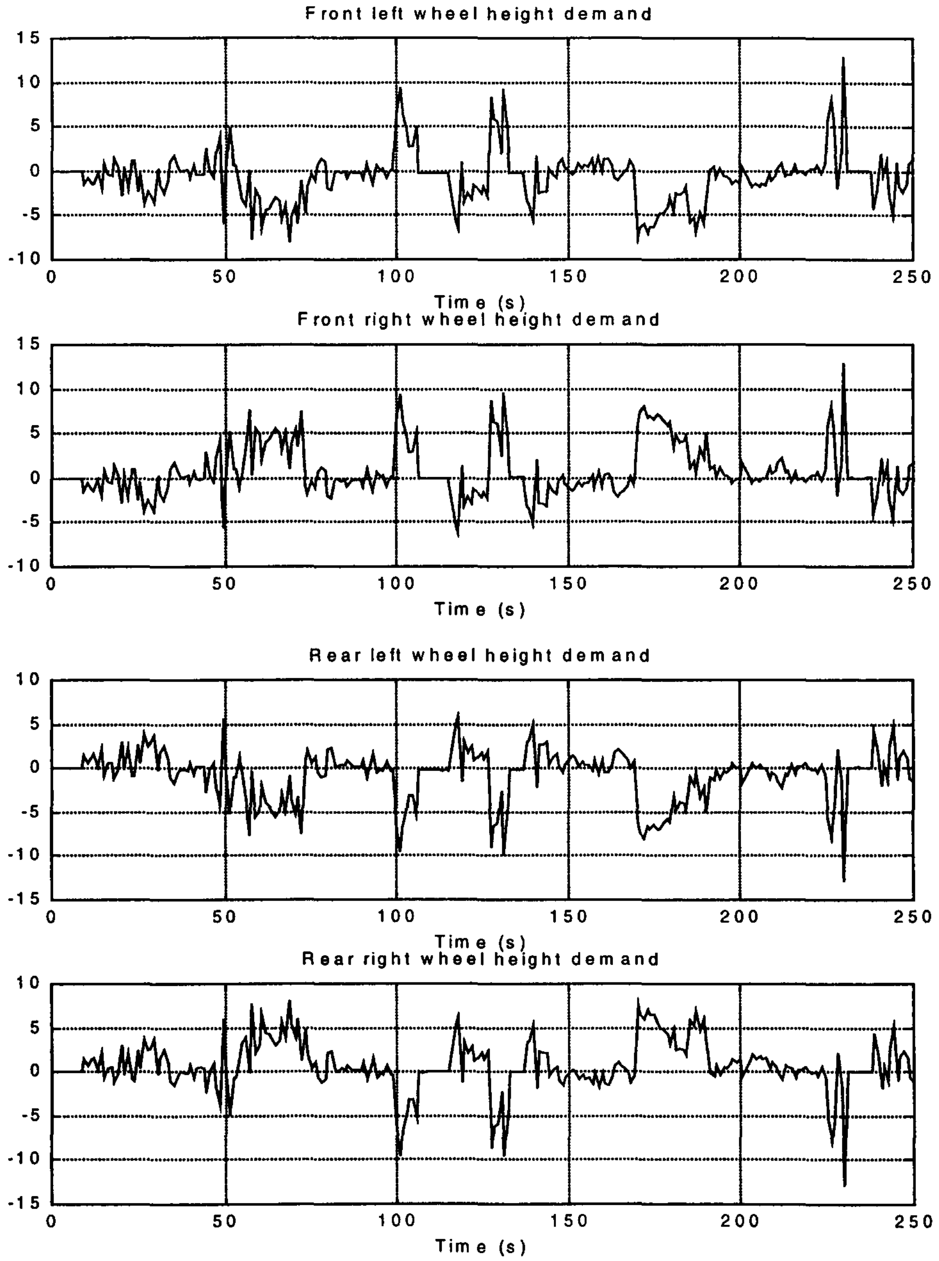


Figure 3.30 Active suspension responses

Figure 3.30 clearly demonstrates the active suspension responses to longitudinal and lateral accelerations. The lateral acceleration plot shows that the vehicle is negotiating two curves starting at approximately 55 and 170 seconds, indicated by the two rises. The active suspension responds by sending commands to raise the two left wheels and lower the two right wheels. This would, in effect, make the four wheel heights even during cornering. As the response to longitudinal acceleration, the longitudinal acceleration plot between 100-150 seconds is considered. During that period, the vehicle is subjected to alternate braking and accelerating, as seen by alternate negative and positive plots, respectively. The active suspension responds by issuing commands to raise the front wheels and lowering the rear wheels when braking, and the opposite when accelerating.

3.4.3 4WS Simulation

The three inputs to the 4WS system, vehicle speed, yaw rate and steering inputs, and its outputs are plotted in Figure 3.31.

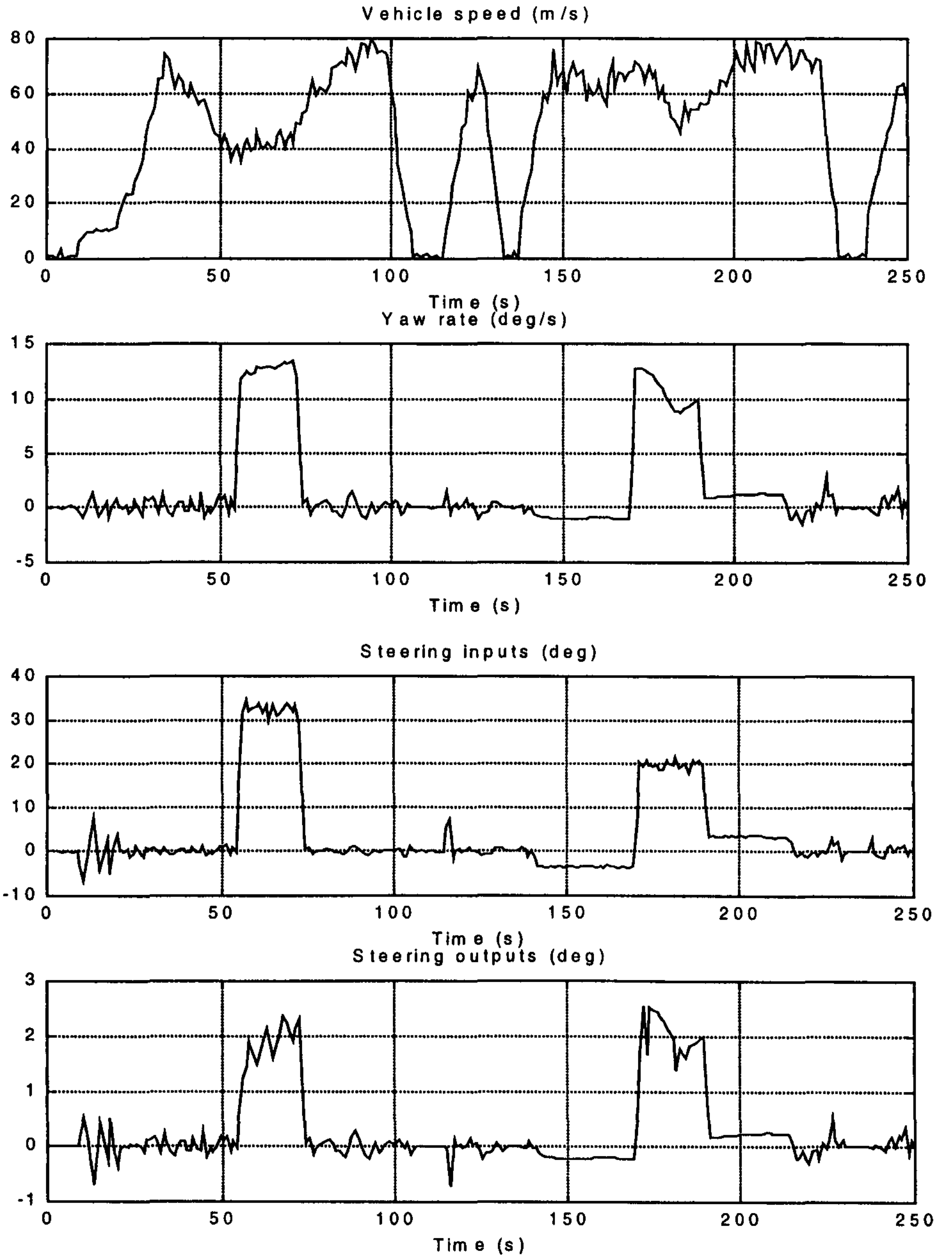


Figure 3.31 4WS inputs and responses

The plot shows the two characteristics of the 4WS system. Starting at approximately 55 and 170 seconds, the front wheels are steered when the vehicle is at medium to high speed in the bends. This can be seen from both the yaw rate and steering input plots. The 4WS responds by steering the rear wheels in the same direction as the front wheel to increase vehicle stability. During low speed drive, seen between 0-25 seconds, the 4WS sends commands to steer the rear wheels in the opposite direction to the front wheels to minimise turning radius.

3.5 Summary

Models of ABS, active suspension and 4WS control systems have been derived in this chapter to match published descriptions of typical systems. In order to be able to exercise these systems, basic road test data has been obtained, from which additional parameters have been synthesised. The simulation shows that the three exemplary systems function as expected. Their information will then be used in other models in this thesis.

3.6 References

1. *Lambourn R* **Braking and Cornering Effects with and without Anti-Lock Brakes** SAE No.940723
2. *Masutomi S, et al.* **Development of ABS and Traction Control Computer** SAE No.901707
3. *Aoyama Y, et al.* **Development of the Full Active Suspension by Nissan** SAE No.901747
4. *Iijima T, et al.* **Development of a Hydraulic Active Suspension** SAE No.931971
5. *Kawakami H, et al.* **Development of Integrated System Between Active Control Suspension, Active 4WS, TRC and ABS** SAE No.920271
6. *Sato H, et al.* **Development of Four Wheel Steering System Using Yaw Rate Feedback Control** SAE No.911922
7. *Jurgen R* **Automotive Electronics Handbook 2nd Edition** McGraw-Hill, 1999
8. *Yokoya Y, et al.* **Integrated Control System Between Active Control Suspension and Four Wheel Steering for the 1989 CELICA** SAE No.901748
9. *Irie N, et al.* **4WS Technology and the Prospects for Improvement of Vehicle Dynamics** SAE No.901167
10. *Bischof H, et al.* **The ECU of A Rear Wheel Steering System** 8th International Conference on Automotive Electronics, 1991
11. *Jun C* **The Study of ABS Control System with Different Control Methods** Proceedings to the Advanced Vehicle Electronics Control (AVEC), 1998
12. *Gillespie T* **Fundamentals of Vehicle Dynamics** SAE, 1992
13. *Lechner D, et al.* **The Actual Use of the Dynamic Performances of Vehicles** IMechE C389/283, 1992
14. *Bauer H* **Automotive Brake Systems** Bosch, 1995
15. *Watanabe K, et al.* **Absolute Speed Measurement of Automobile from Noisy Acceleration and Erroneous Wheel Speed Information** SAE No.920644
16. *Milliken W, et al.* **Race Car Vehicle dynamics** SAE, 1995

CHAPTER 4

VEHICLE ELECTRONIC ARCHITECTURES UNDER CONSIDERATION

The distributed wheel controller and four other automotive dynamic control architectures were chosen for modelling and comparison purposes. The study and comparison between these architectures, with regards to their feasibility, advantages and disadvantages in the aspects explained in Chapter 1, will be described in the following chapters. The other four architectures were selected because of their current popularity or potential for the future as suggested in literature. They will each be described in this chapter.

4.1 Conventional Centralised Architecture (*Architecture 1*)

Before the introduction of networking, this was the only architecture of car electronics. The architecture is still the most commonly applied architecture at present, especially in medium and small-sized cars. It is, therefore, chosen to be the basic model for comparison. Electronic control systems here contain centralised ECUs. The ECUs are linked with corresponding sensors and actuators via dedicated wire links. Each type of data transfer between ECUs are also through these individual links. The model of the conventional centralised architecture is shown in Figure 4.1.

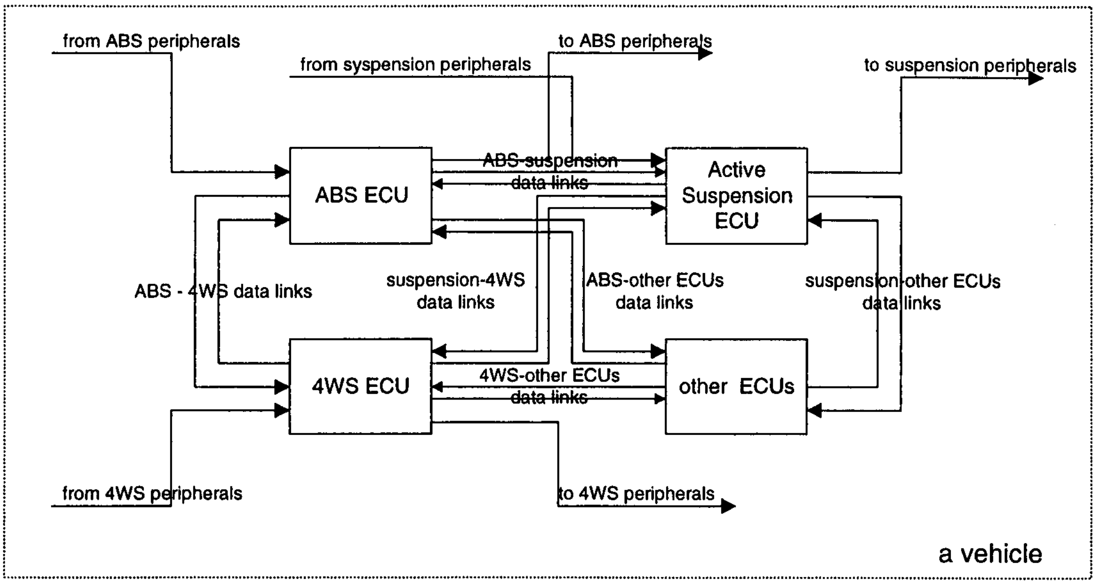

from ABS peripherals
to ABS peripherals
from syspension peripherals
to suspension peripherals
ABS ECU
ABS-suspension data links
Active Suspension ECU
ABS - 4WS data links
suspension-4WS data links
ABS-other ECUs data links
suspension-other ECUs data links
4WS ECU
4WS-other ECUs data links
other ECUs
from 4WS peripherals
to 4WS peripherals
a vehicle

Figure 4.1 Conventional centralised architecture

4.2 Conventional Centralised with Limited CAN Interaction Architecture (*Architecture 2)*

This architecture is currently gaining popularity along with networking, and is believed to be the most common system of the near future. It represents how networking is applied in modern vehicles. In this architecture, all the electronic systems are of conventional structures, each with an ECU at the control centre dictating its peripheral hardware. A network bus is introduced to replace hard wires for data sharing among ECUs. The system is as shown in Figure 4.2.

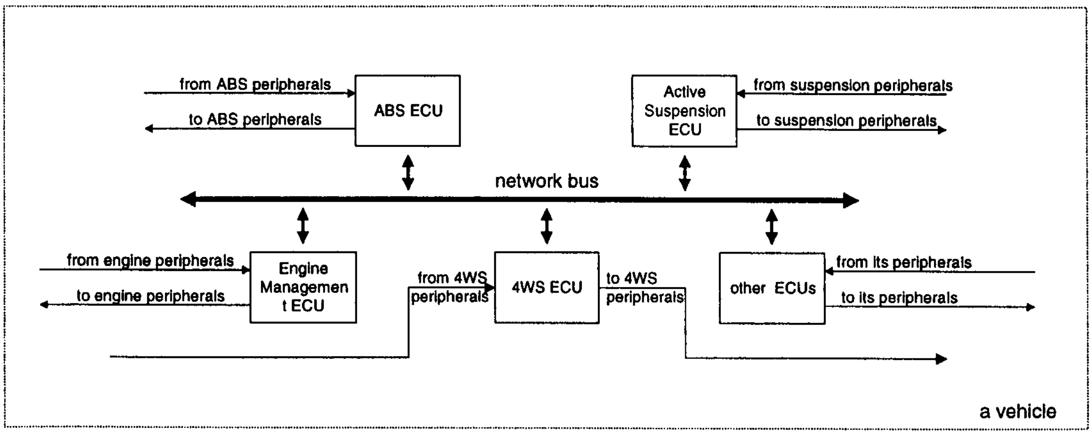


Figure 4.2 Conventional centralised with limited CAN interaction architecture

4.3 Total Centralised Architecture (*Architecture 3)*

This architecture follows an idea of integration to reduce wiring and number of devices, by combining a number of ECUs into a single unit. In practice, it started by integrating engine management system and transmission control ECUs into one unit. It could expand further to combine other dynamic control ECUs which are in the same geographical area [1]. This is an alternative architecture for the future. The combined ECU would control all the related systems and may share data with other ECUs via networking. A diagram of the architecture is as shown in Figure 4.3.

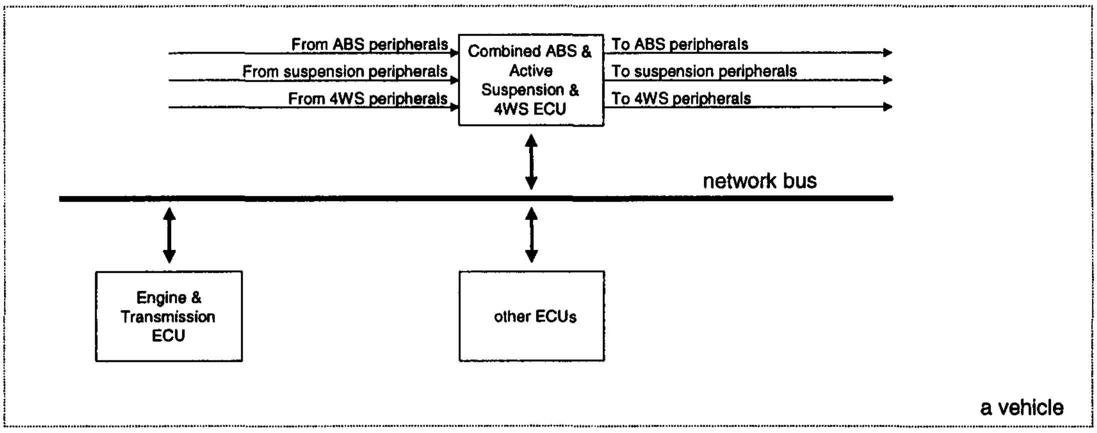

From ABS peripherals
From suspension peripherals
From 4WS peripherals
Combined ABS & Active Suspension & 4WS ECU
To ABS peripherals
To suspension peripherals
To 4WS peripherals
network bus
Engine & Transmission ECU
other ECUs
a vehicle

Figure 4.3 Total centralised architecture

4.4 Conventional Centralised with Functional Integration Architecture (*Architecture 4*)

This architecture is based on the existing *Architecture 2* with enhanced vehicle control performance. The concept initiates interests and research in many ECU developers and car manufacturers [2-5]. Each control system has its centralised ECU to command its operation. There exist smart sensors and actuators in each system, which contact the ECU via a network bus. By doing so, more data from each system is available to other ECUs. There are also more interactions between ECUs. This enables more functional integration among control systems, such as combined brake, suspension and steering control during cornering. The system diagram is displayed in Figure 4.4.

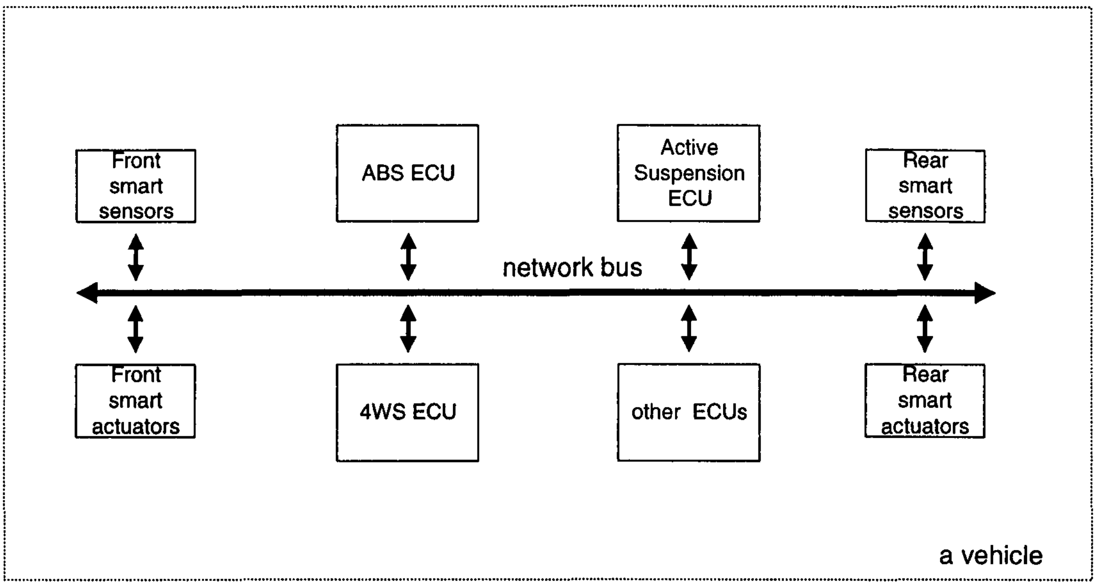


Figure 4.4 Conventional centralised with functional integration architecture

4.5 Distributed Wheel Controller Architecture (*Architecture 5*)

In this architecture, a combined ABS, 4WS and active suspension control ECU for each wheel is situated near to that wheel. Each Wheel Controller (or Distributed) ECU would also be integrated with the wheel sensors and actuators to form a single unit. The Central ECU would provide some share sensor data and monitor the Wheel Controller ECUs. The four wheel controllers are linked to other ECUs by a network bus as shown in Figure 4.5.

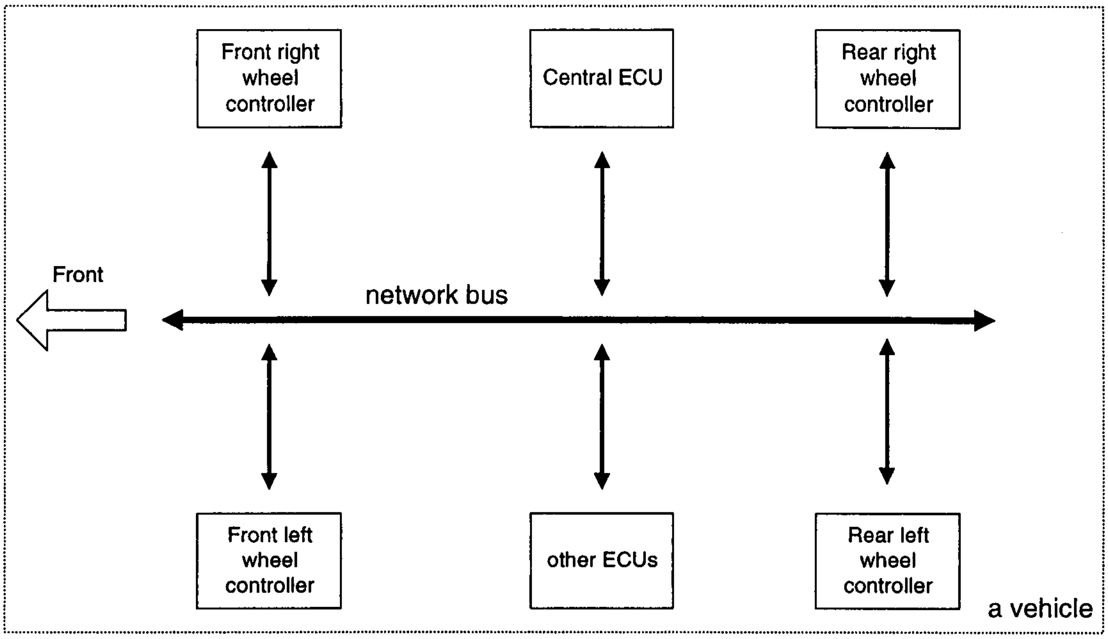

Front right wheel controller
Central ECU
Rear right wheel controller
Front
network bus
Front left wheel controller
other ECUs
Rear left wheel controller
a vehicle

Figure 4.5 Distributed wheel controller architecture

4.6 References

1. *Emaus B* **Aspects and issues of multiple vehicle networks** SAE No. 950293
2. *Kiencke U* **Integrated Vehicle Control Systems** IFAC Intelligent Components for Autonomous and Semi-autonomous Vehicles, 1995
3. *Schmidt E, et al.* **Required elements of integrated vehicle control systems** SAE No.901170
4. *Wallentowitz H* **Scope for the integration of powertrain and chassis control systems: Traction control - All-wheel drive - Active suspension** SAE No.901168
5. *Tanaka H, et al.* **Development of a Vehicle Integrated Control System** ImechE C389/220, 1992

CHAPTER 5

ELECTRONIC ARCHITECTURE FEASIBILITY STUDIES ON HARDWARE REQUIREMENTS AND CAN MESSAGE DELAY

5.1 ECU Hardware and Performance Requirements

This chapter is intended to address the feasibility of the five alternative vehicle electronic architectures described earlier; specifically the requirements on the microcontrollers that will control the ECUs. Computing performance and electronic interface specifications are estimated for these. In addition, the data transfer time associated with CAN implementation on different architectures is studied. These two studies provide more details on electronic characteristics of these architectures, as well as indicating their relative feasibility for implementation.

The work in this thesis concentrates upon electronics hardware aspects of the vehicle electronic architectures. Since ECUs and especially microcontrollers are at the hearts of these systems, their characteristics in particular should be examined. Furthermore some architectures such as *Total Centralised* (*Architecture 3*) have not been previously implemented, hence it is useful to verify its possibility and evaluate its likely contents. An important characteristic of CAN is its timing, which is crucial to these safety related control systems. The high speed (class C) intra and inter system data transfer is required to be within specified time limits, so that up to date data can be used for vehicle control tasks.

5.1.1 The Vehicle System Design Process

ECU requirements specification setting is the first part of the ECU development process. This stage is followed by electronic architecture design, implementation and test [4]. In the conventional way of ECU development up till now, the functional specifications including the control algorithms and time and accuracy requirements of the ECUs are specified by a development team [5]. These specifications are passed on to the ECU developer, who selects the ECU hardware including a microcontroller, and programs it to perform the specified functions. The ECU hardware as well as the functional design are experimentally tested and corrected until the development team is satisfied. In the new ECU development process, its functions are tested by simulation until satisfied before a pilot ECU is constructed and tested. This enables development time saving [6].

In both conventional and recent ECU development processes, the hardware specifications including those of the microcontroller, are determined after the details of the ECU functions and requirements are made. These complete ECU functional details, which includes all the I/O signals required, control and diagnostic algorithms, are enough to specify the hardware as well as developing the software of the ECU.

In this thesis, detailed ECU specifications are not available for the existing electronics architectures and are non-existent for the future and proposed architectures. The only available information is the functional description of each control system, which can be obtained from the literature.

In order to estimate the required hardware specifications of the ECUs of these future electronic architectures, an initial estimate is made based on the performance of existing control ECUs. Thus, if the accuracy of the prediction can be determined from considering a known ECU, and any necessary correction factor established, this same factor can then be applied to calculations for future systems.

5.1.2 Hardware Performance Estimation of Known ECUs

As mentioned above, the objective of work described here is to estimate the time taken to perform a specific control task by a known ECU. This is carried out by studying the function of the ECU and derive a program pseudo code to execute the tasks. Since the type of microcontroller used is known, so is its timing specifications for each op code. Hence the total time the microcontroller requires to perform the tasks can be estimated. An actual response time can also be measured experimentally. The two results are compared and errors identified. The technique can then be used to estimate the performance required of other control ECUs of different electronic architectures, taking the same errors into account.

5.1.2.1 Response Time Calculation

Different microcontrollers require particular numbers of operating cycles for each instruction. To calculate the time they take to perform a task, the total number of times each instruction is to be executed is counted. These sums are the universal platform for microcontroller response time calculation. They are multiplied by the number of execution cycles of the corresponding commands of a microcontroller. The sum of these products is a total number of execution cycles that a specific microcontroller requires to do the task. The execution time is then the product of the total number of execution cycles and the execution cycle time. Equation 5.1 represents the calculation process.

$$T = T_c . t_{CYC} \qquad (5.1)$$

Where $T_c = (n_1.c_1 + n_2.c_2 + \ldots + n_p.c_p)$

T	- control execution time
T_c	- total execution cycle count
n_x	- number of times an instruction is to be run
c_x	- number of execution cycles a command takes
t_{CYC}	- a microcontroller execution cycle time

The two ECUs studied in this work were an air suspension control and a cruise control ECUs. They were chosen due to the hardware availability and availability of design documentation, and were slightly less complex (in term of code) than the future systems.

The next section will describe the work on both ECUs.

5.1.3 Cruise control ECU Experiment

The ECU examined was the cruise control ECU part no. 5GA 004 397 developed by Hella, fitted to Range Rover vehicles. The microcontroller used in the ECU was a Toshiba 8-bit equivalent of the Intel 80C49, running at 8 MHz [3].

From the Range Rover Workshop Manual, the cruise control system functions are as follows [1]:

- when SET is pressed and the cruise control is not in the cruising mode, the cruise speed will be set and the cruise control will start to maintain vehicle speed at that constant set speed. The cruise speed will be set provided that the vehicle speed exceeds 28 mph.

- when SET is pressed during the cruising mode, the cruise control will accelerate the vehicle by operating the motor that opens the throttle. Once the SET button is released, speed increase is stopped, and the cruise control sets the cruise speed to the current vehicle speed.
- when RES is pressed during cruising mode, the cruise control operation will be disengaged
- when RES is pressed again, the cruise operation resumed, with previously set speed restored.
- at any time during its operation, the cruise control will be disabled when either a stop lamp switch is on, or the vehicle speed drops below 28 mph.
- other driver's controls such as accelerator and clutch pedals depression, non forward gearbox, and out of range engine speed can also disable the cruise system, but they are monitored by the Body Electrical Control Module (BECM). Once any of these conditions occurs, the BECM will simply cut the cruise control power supply. Hence they are outside the scope of the cruise control.

This moding is shown in a state transition diagram in Figure 5.1.

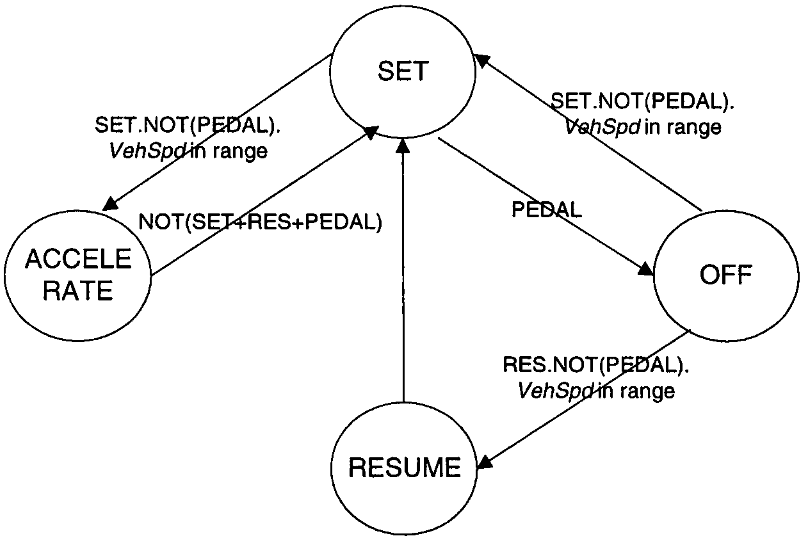


Figure 5.1 Cruise control state transition diagram

SET cruise control sets current speed as set speed and maintains constant set speed

RESUME cruise control restores previously installed set speed

ACCELERATE cruise control accelerates while SET is pressed

OFF cruise control stops vehicle speed control

5.1.3.1 Throttle Actuator Operations

The throttle actuator consists of a vacuum pump motor and solenoid. Their operations are [1]:

- Open throttle when the pump motor operates, air pressure is applied to the actuator to open the throttle.
- Constant throttle during operation, the normally closed solenoid valve is closed to keep the throttle angle constant.
- Close throttle the solenoid valve is released when the throttle is to be closed, letting the air out of the cylinder.

The operation is illustrated in Figure 5.2.

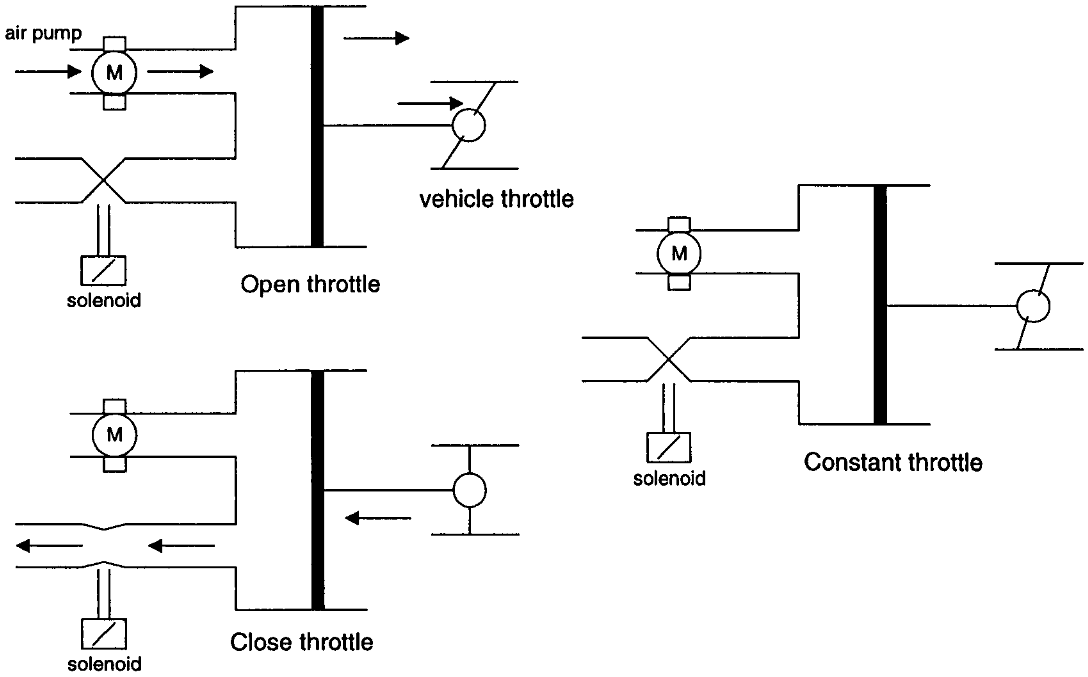


Figure 5.2 Cruise control motor and solenoid operations

5.1.3.2 Estimation Method

The parameter that was chosen as the basis of the comparison between theoretical and practical results, was the systems response time to the two cruise control switch inputs, 'SET' and 'RES'. These two switches are used to initiate vehicle cruise control. The comparison of the two results would show what level of inaccuracies the method incurred.

5.1.3.3 Theoretical Response Time

The response time of a cruise control unit, to the two stimuli, can be estimated by looking into the program instruction level of the microcontroller. The estimate is made by first studying the cruise control functions derived from the data sheet in Figure 5.1, and then drawing the equivalent program structure. It is acknowledged that from the functions, a variation of program structures could be drawn. However, it is assumed that the most calculation intensive, hence most processor time consuming program part, which is the vehicle speed control algorithm is essentially the same. Variations in other program parts are hence believed to have little effect on the overall response time. Having completed this estimated program structure, the equivalent machine code instructions can be deduced. The individual instruction execution times, provided in the microcontroller data sheet, can then be used to estimate the total time taken for program execution, and hence the response time. The assumption is made that all moding is handled by the microcontroller i.e. the surrounding logic performs only very simple operations, such as inverting/combining signals, or storing results (latches), and that the estimated program structure matched that which was actually used.

From the cruise control functions written earlier, the program flowchart is deduced as shown in Figure 5.3. The coloured lines are response time measurement guidelines, and processes in bold indicate changes in states of input or output, hence detectable.

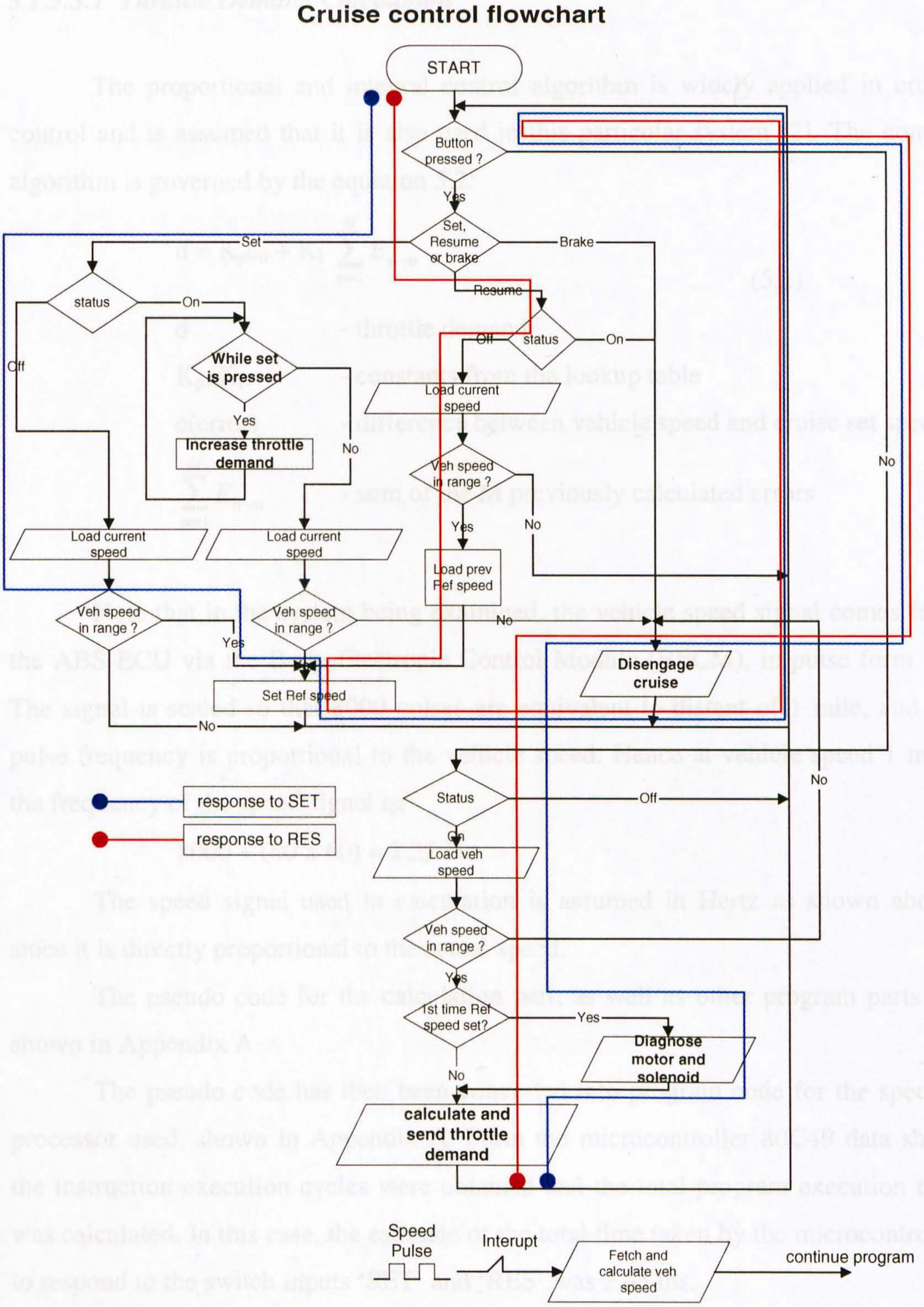


Figure 5.3 Cruise control operation flowchart

From Figure 5.3, the calculation intensive and hence the most time consuming section is the throttle demand calculation, which involves finding the speed error and applying the speed control algorithm. This is expected to take significantly more of microcontroller time than others. It will thus be considered in more detail.

5.1.3.3.1 Throttle Demand Calculation

The proportional and integral control algorithm is widely applied in cruise control and is assumed that it is also used in this particular system [2]. The control algorithm is governed by the equation 5.2:

$$d = K_p e_n + K_I \sum_{m=1}^{M} E_{n-m} \qquad (5.2)$$

d	- throttle demand
K_p, K_I	- constants from the lookup table
e(error)	- difference between vehicle speed and cruise set speed
$\sum_{m=1}^{M} E_{n-m}$	- sum of the M previously calculated errors

Note that in the system being examined, the vehicle speed signal comes from the ABS ECU via the Body Electronic Control Module (BECM), in pulse form [1]. The signal is scaled so that 8000 pulses are equivalent to distant of 1 mile, and the pulse frequency is proportional to the vehicle speed. Hence at vehicle speed 1 mph, the frequency of the speed signal is:

$$8000 \div (60 \text{ x } 60) = 2.22 \text{ Hz}$$

The speed signal used in calculation is assumed in Hertz as shown above, since it is directly proportional to the actual speed.

The pseudo code for the calculation part, as well as other program parts are shown in Appendix A.

The pseudo code has then been converted into program code for the specific processor used, shown in Appendix A. From the microcontroller 80C49 data sheet, the instruction execution cycles were obtained and the total program execution time was calculated. In this case, the estimate of the total time taken by the microcontroller to respond to the switch inputs 'SET' and 'RES' was 2.65 ms.

5.1.3.4 Practical Measurement of Response Time

A set of practical measurements were then made using a real cruise control ECU to reproduce the theoretical estimates as closely as possible.

5.1.3.4.1 Circuit Connection

The cruise control circuit was then connected to a simple test jig for measurement of the same timing parameters. The connection was as shown in Figure 5.4.

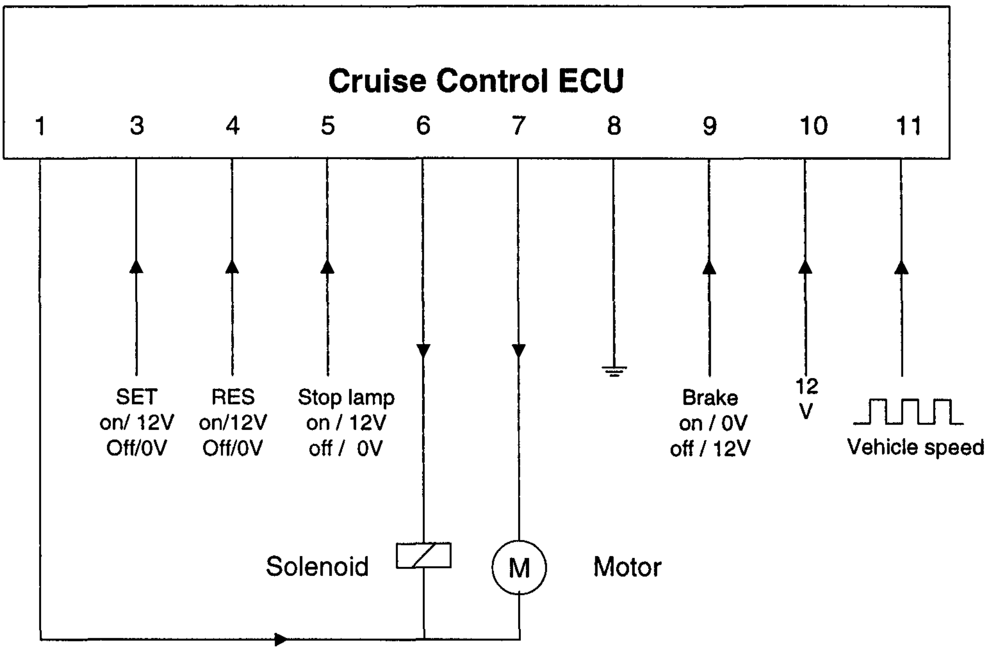


Figure 5.4 Cruise control experiment circuit connection

The equipment used in this experiment was as follows:

- To observe if the motor and solenoid were being switched on, they were replaced by LEDs
- toggle switches were connected as brake and stop lamp switches
- 'SET' and 'RES' were controlled by button switches, fed from a 12V supply
- vehicle speed signal was represented by a square wave from the signal generator.
- the logic analyser and a digital oscilloscope were used as measurement tools.

5.1.3.4.2 Identification of Internal Connections

In order to be able to identify the response of the microcontroller, rather than any external filtering or timing circuits, the ECU was opened and an inspection was made of the microcontroller connections. The results were:

SET (ECU pin 3)	corresponds to 80C49 pin 12 (BUS bit 0)
RES (ECU pin 4)	corresponds to 80C49 pin 17 (BUS bit 5)
Motor (ECU pin 7)	corresponds to 80C49 pin 32 (Port 1 bit 5)
Solenoid (ECU pin 6)	corresponds to 80C49 pin 28 (Port 1 bit 1)
Veh Speed (ECU pin 11)	corresponds to 80C49 pin 6 (Interrupt)

Figure 5.5 shows the internal connection diagram.

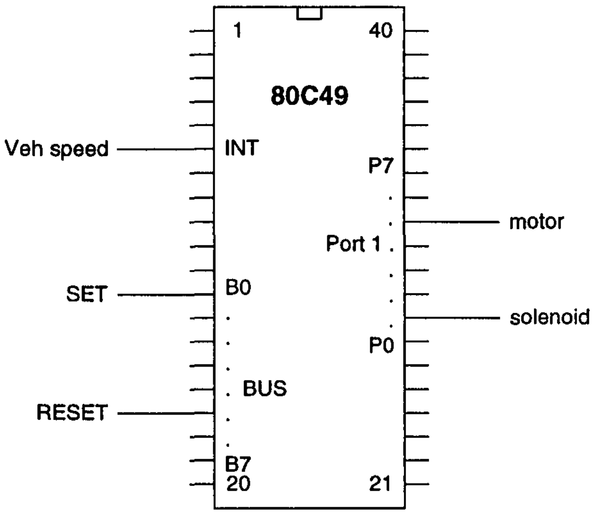


Figure 5.5 Cruise control microcontroller connections

The vehicle speed input thus causes an interrupt to the controller program on every pulse. The clock speed of the microcontroller was verified as being 8 MHz.

5.1.3.4.3 Measurements

Measurements were taken of the response time taken for the output signals to motor and solenoid to change state after one of SET and RES buttons was pushed.

The measurements were taken as being the time delay between button pressed and a change in motor or solenoid output, according to the coloured guidelines in the flowchart shown in Figure 5.3. The results were compared to the prediction model. For each measurement, the timing was repeated over 30 samples to ensure consistency, and any apparent dependency on vehicle speed was investigated. It was, however, believed that despite the use of interrupt to measure vehicle speed, the effect of vehicle speed on response time would be minimal. This is because from the pseudo code in Appendix A, the vehicle speed calculation would take less than one hundredth of the predicted response time to switch inputs.

5.1.3.4.3.1 Response Time when SET (blue line on Figure 5.3)

The response time was measured between the point that 'SET' is pressed to store and maintain the current vehicle speed, and when the control signals to motor and solenoid change state.

The measurement was made when 'SET' button was pressed to set the cruise reference speed for the first time.

The logic analyser was set to trigger the change in SET button signal. Its internal clock was set at 5 kHz. The waveforms are displayed in Figure 5.6.

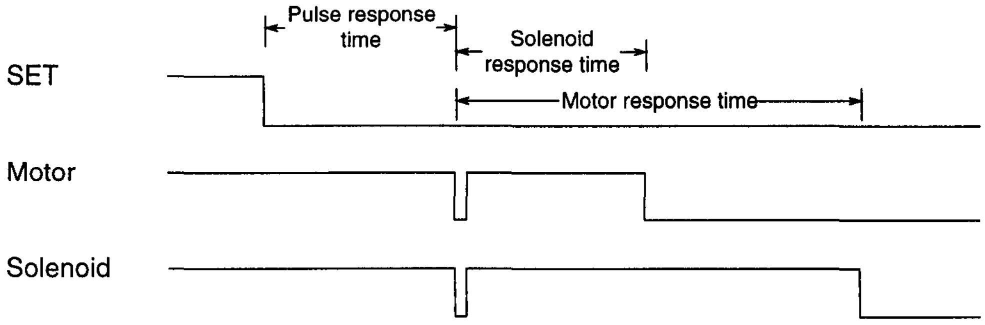


Figure 5.6 Waveforms when SET is pressed

It can be seen that a short pulse is applied to each motor and solenoid initially, followed by full actuation several milliseconds later.

The hypothesis is that solenoid and motor are diagnosed by the ECU by sending a test signal, a short negative-going pulse, to check the two actuators. If they pass this check, they are then turned on by the falling edge a short time later. The three periods of time were measured at different cruise set speed and plotted as illustrated by Figure 5.7, 5.8 and 5.9.

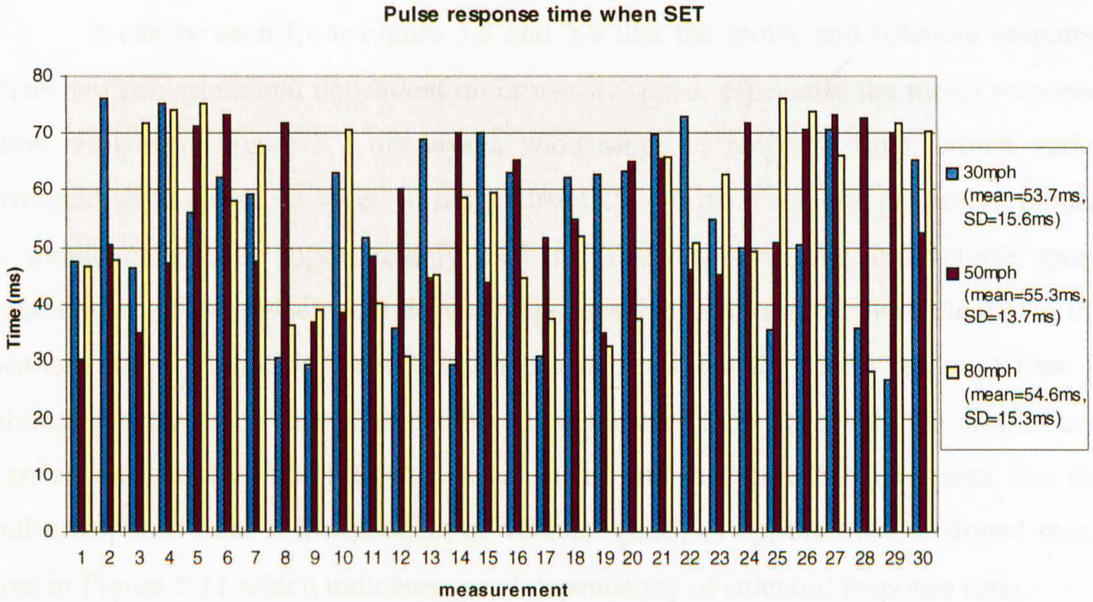


Figure 5.7 Pulse response time to SET command

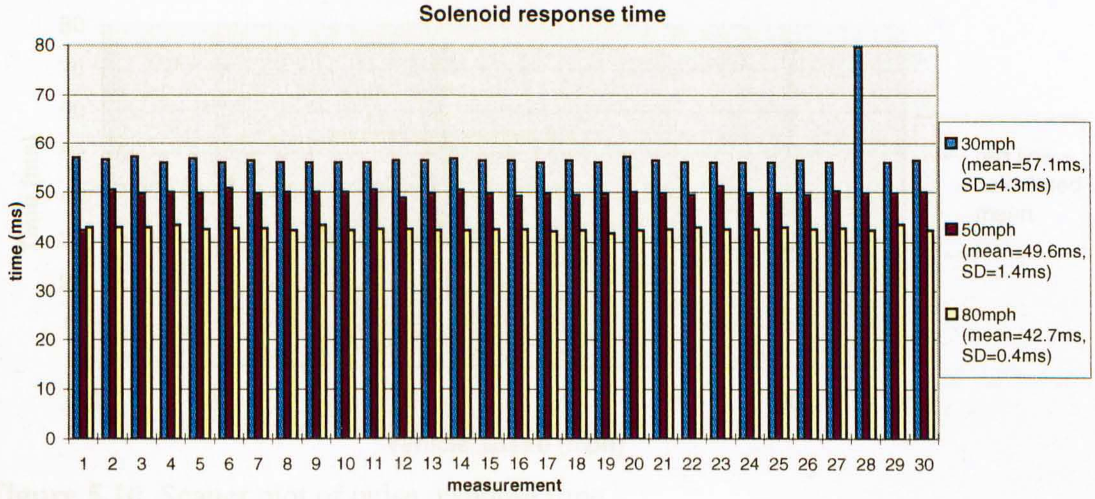


Figure 5.8 Solenoid response time

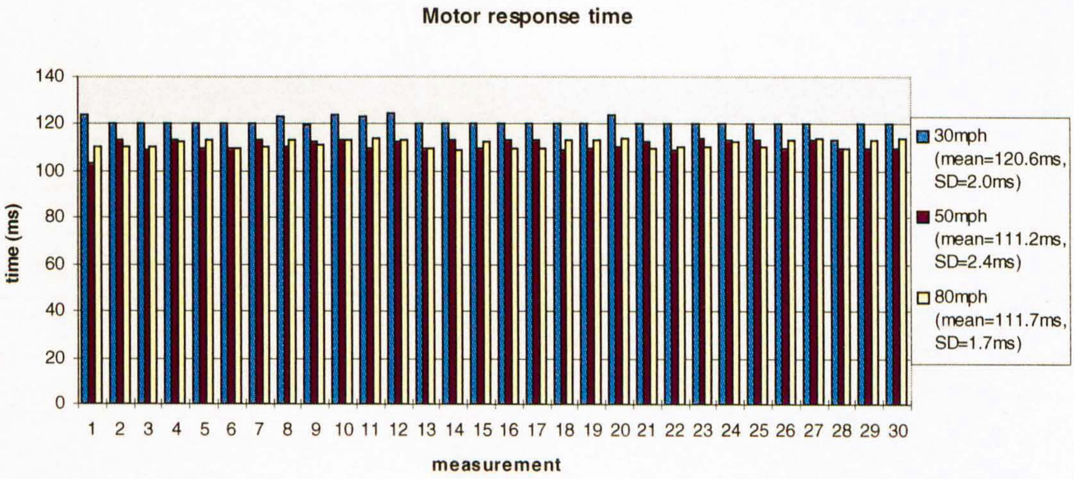


Figure 5.9 Motor response time

It can be seen from Figure 5.8 and 5.9 that the motor and solenoid response times are consistent and dependent on cruise set speed, especially the motor response time. However, Figure 5.7 displays a wide range of response time, which varies irrespective to cruise set speed. It ranges from 28 - 76 ms. From the prediction model, it should only takes approximately 2.65 ms. The response time and vehicle speed relationship of the pulse and solenoid responses can be observed more clearly in the scatter plot of Figure 5.10 and 5.11. It can be seen that the motor response time is almost exactly twice that of the solenoid, suggesting the two events are similar and carried out serially. The relatively level mean line in Figure 5.10 suggests that the pulse response time is independent of vehicle speed, as opposed to the sloped mean line in Figure 5.11 which indicates speed dependency of solenoid response time.

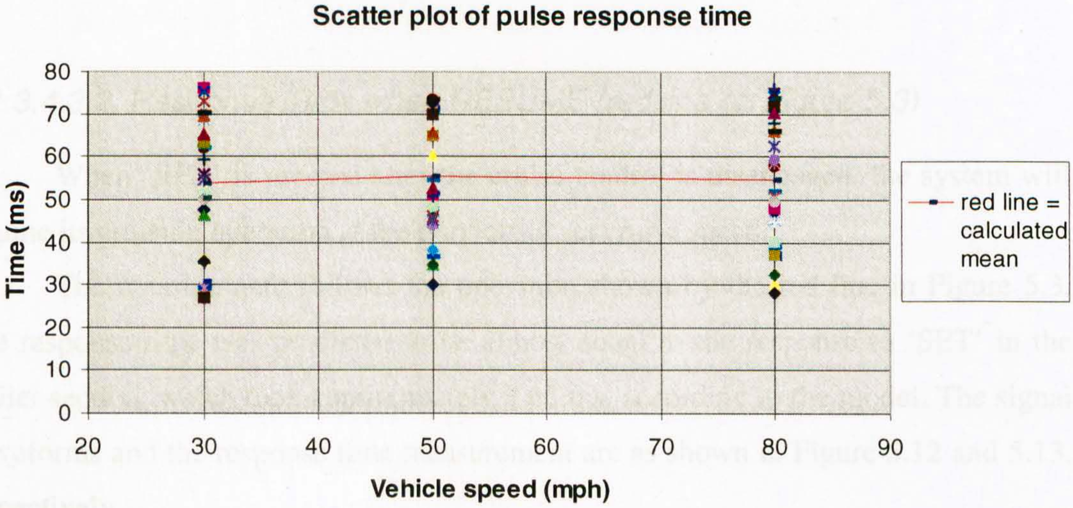


Figure 5.10 Scatter plot of pulse response time

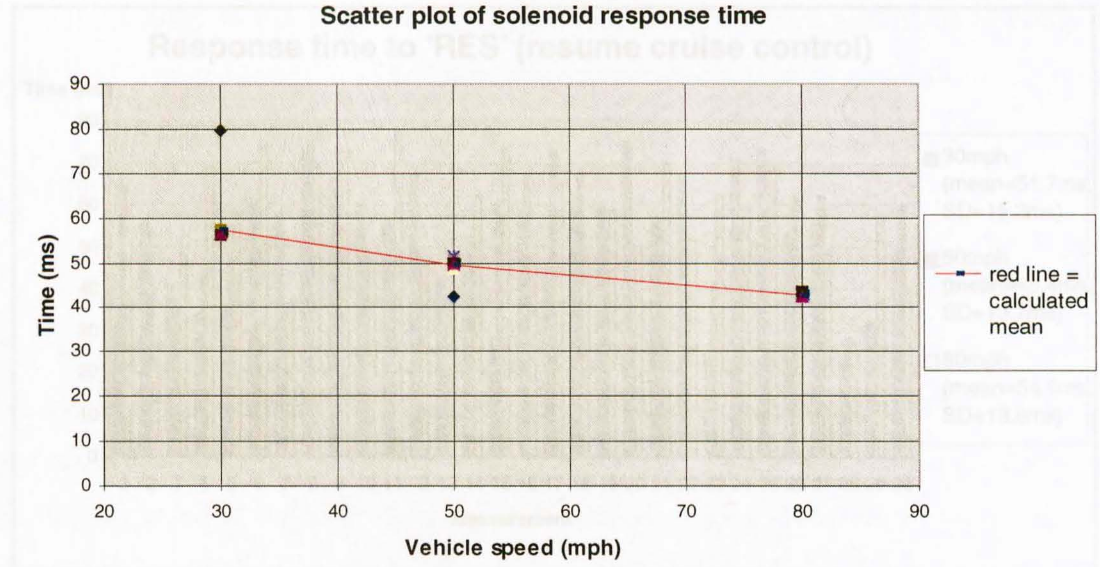


Figure 5.11 Scatter plot of solenoid response time

5.1.3.4.3.2 Response Time when RESUME (red line on Figure 5.3)

When 'RES' is pressed after the cruise control is disengaged, the system will resume its cruising operation at the previously set cruise speed.

The measurement follows the operation shown by the red line in Figure 5.3. The response time was predicted to be almost equal to the response to 'SET' in the earlier section, which took approximately 2.65 ms, according to the model. The signal waveforms and the response time measurement are as shown in Figure 5.12 and 5.13, respectively.

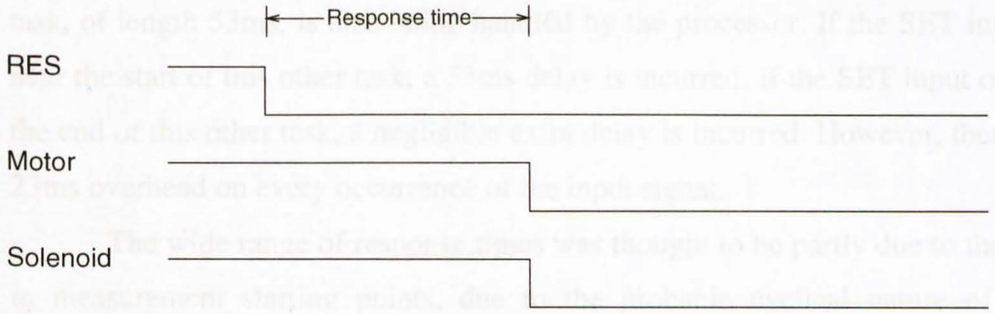


Figure 5.12 Signal waveforms when 'RES' is pressed

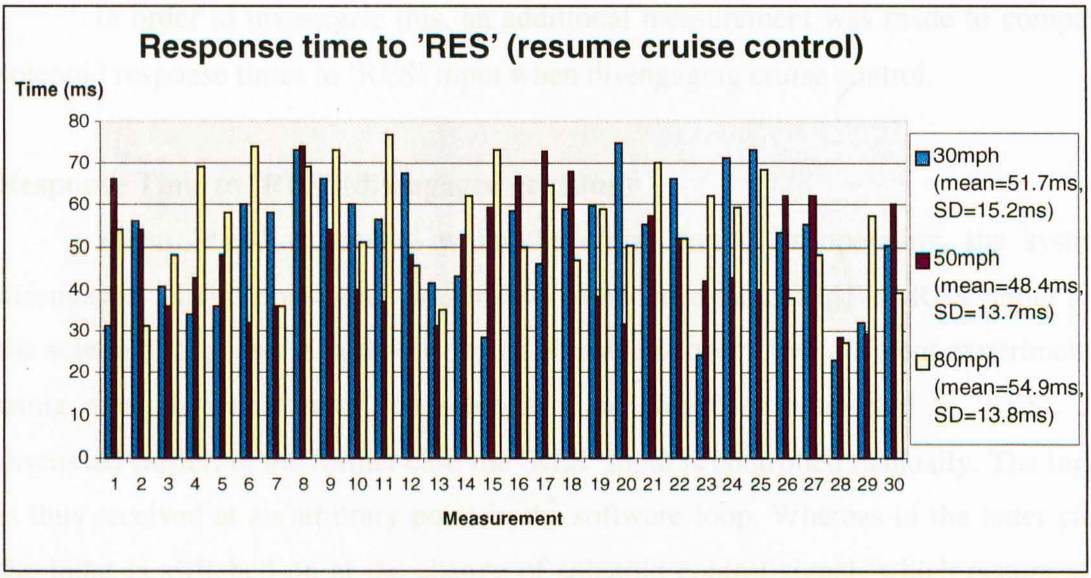


Figure 5.13 Response time to SET (after cruise disengaged)

From Figure 5.13, the response time is inconsistent, ranging from 23 to 76ms. It is much more than that anticipated by the model at 2.65ms. The vehicle speed, as predicted, has virtually no effect on the response time, since the average response times measured at different vehicle speed were within 15% of one another, with the overall average response time of 52 ms. However, these comparable averages indicate that the program loops to perform these tasks are comparable in terms of time taken, as it would be expected from the model. The average response time is also comparable to the average pulse response time to 'SET' (Figure 5.7). This similarity of response times, again, was expected from the model.

From the range of measured response times, it is likely that a non-interruptible task, of length 53ms, is also being handled by the processor. If the SET input occurs near the start of this other task, a 53ms delay is incurred. If the SET input occurs near the end of this other task, a negligible extra delay is incurred. However, there is still a 23ms overhead on every occurrence of the input signal.

The wide range of response times was thought to be partly due to the variation in measurement starting points, due to the probable cyclical nature of software. Although it was assumed to start from the top of the flowchart in Figure 5.3, it could in fact start anywhere in the calculation loop. From the flowchart, the state of the button is monitored once in each loop. If it is pressed while other tasks are performed, the controller will not detect it until it completes all the loop functions and looks at the button.

In order to investigate this, an additional measurement was made to compare solenoid response times to 'RES' input when disengaging cruise control.

Response Time to 'RES' (disengaged cruising)

When 'RES' is pressed while the cruise control is operating, the system disengages. The response time was measured between the change in 'RES' input and the solenoid response. A comparison can be made between two different experiments, using manual switch input and solenoid signal feedback connected to 'RES'. As discussed earlier, in the former case the 'RES' input is controlled manually. The input is thus received at an arbitrary point in the software loop. Whereas in the latter case the input is switched on at the change of solenoid control signal, which occurs at a certain point in software. This comparison between the two cases can indicate the effect of the variation in the measurement starting points.

The response time to 'RES' controlled manually is shown in Figure 5.14.

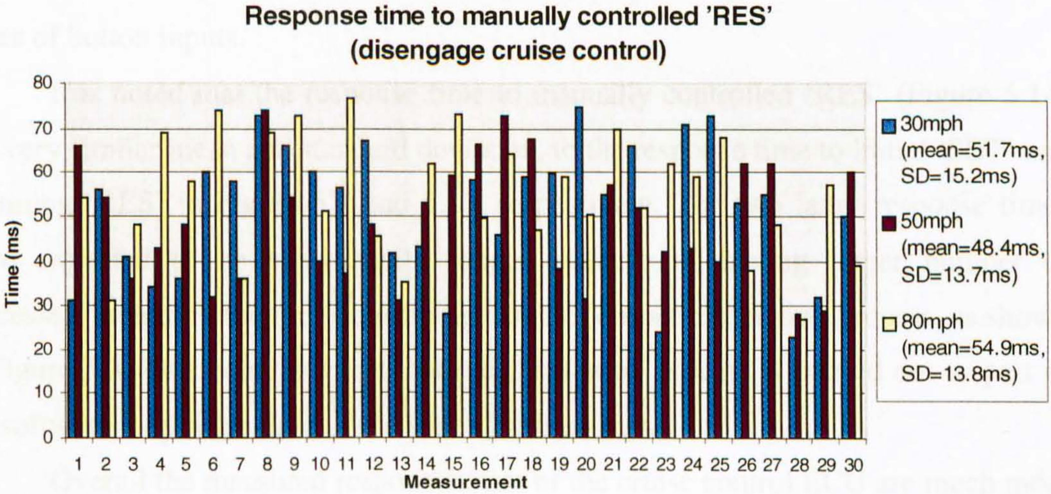


Figure 5.14 Response time to manually controlled 'RES' (disengage)

The solenoid control feedback to 'RES' was done by first connecting it to a simple BJT switching circuit to invert the output, since the solenoid control and 'RES' input signals had opposite on/off status. The inverted solenoid control signal was then connected to the 'RES' input pin.

The response time to the solenoid signal feedback controlled 'RES' is shown in Figure 5.15.

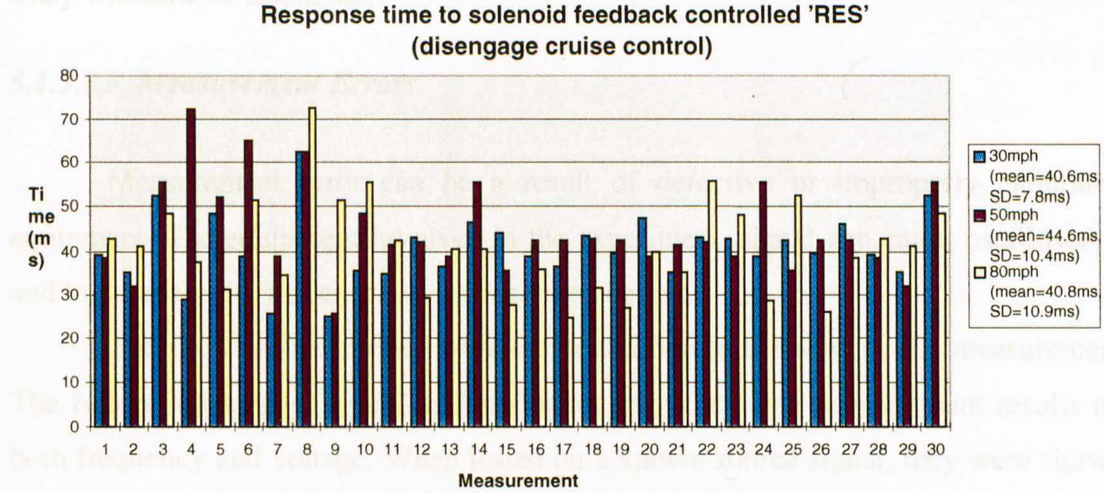


Figure 5.15 Response time to 'RES' (disengage), controlled by solenoid signal

From Figure 5.15, the range of readings obtained are smaller than those when using manual 'RES' switch, as seen from the reduction in standard deviation (average reduction of 5.4ms). It also has 9.5ms shorter average time. This demonstrates that there is a certain point in the software, at which the microcontroller monitors the states of button inputs.

It is noted that the response time to manually controlled 'RES' (Figure 5.14) has very similar mean and standard deviation, to the response time to initial 'SET' and resuming 'RES' in Figure 5.7 and 5.13, respectively. The two latter response times were expected to be significantly longer because of having larger number of processes, including the time consuming actuator output calculation process, as shown in Figure 5.3. This could suggest that the calculation process is carried out as part of the software loop regardless of the state of switch input.

Overall the measured response times of the cruise control ECU are much more than the predicted ones. This can be attributed to several factors, as will be discussed in the analysis.

5.1.3.5 Analysis

The measured results above are different from those predicted by the predicting model of the equivalent ECU operations. There are three possible causes to differences between measured and predicted results. These can be categorised as:

- measurement errors
- external effects (test jig)
- internal effects (ECU)

They will now be discussed.

5.1.3.5.1 Measurement Errors

Measurement error can be a result of defective or improperly calibrated equipment. The equipment involved in the experiment, signal generator, oscilloscope and logic analyser, was checked, as described below.

The oscilloscope probes were calibrated before the start of the measurement The two oscilloscopes, digital and analogue, gave the same measurement results on both frequency and voltage. When tested on a known source signal, they were shown to be accurate to within 3% on the test signal.

Next the signal generator frequency calibration was checked, by using an oscilloscope to monitor frequency of a signal generated. The frequency range is within ± 1% of the setting, which is considered acceptable. The voltage control knob was also found to be working in order.

Finally, the logic analyser was examined by measuring the cruise control response time, using both a digital oscilloscope and the analyser. The two results were within ± 1% of each other. They are hence considered sufficiently precise.

From the above examination, measurement error can then be assigned a maximum value of 3%.

5.1.3.5.2 External Effects

External effects include a variation in performance of other components associated with the microcontroller and cruise control ECU.

During the experiment, all the components and signals connected to the ECU, such as vehicle speed signal and switch status, were controlled so that their conditions were identical for repeated measurements.

The signal generator accuracy has been confirmed, but another possible effect may be from the switches. Mechanical switches have a contact bounce time, when pressed, before they settle to one state. This bounce time was measured for the switches used and was found to be between 130 to 650 μs. This time duration was negligible compared to the response time of tens of milliseconds.

A relay in the ECU also has the effect on the response time but its maximum delay is measured to be approximately 1 ms. This is again, negligible compared to the duration of response time. Hence the major effect is unlikely to lie in this area.

5.1.3.5.3 Internal Effects

It is very likely that the unknown program in the microcontroller has a significant effect on the response time of the cruise control. The wide range of response time obtained, indicates a possibility that the microcontroller was executing a certain program loop when SET or RES was pressed. It would complete its current task before responding to the input. The response time obviously depends on at which point in the loop, a button is being pressed. If it is near the start of the loop, the response will be longer. On the other hand, if it is near to the end, the microcontroller quickly finishes the loop and thus gives fast response. The experiments on manually and solenoid signal feedback controlled 'RES' switch demonstrates this possibility.

The underestimated actuator control signals may result from the different control algorithm adopted, or it could also be different calculation methods employed to get the results. The resolution of inputs and outputs also has effect on calculation time.

The model does not take the self and peripheral device diagnostic times into account simply because the methods used is unknown and entirely up to the software designer. If there were such tasks, they would take a significant amount of the microcontroller time.

The software may be designed to allow for mechanical delays in the cruise control system.

Although the switch bounce itself is very short, it is highly possible that the software allows ample time for switch debounce for safety purpose. It is found that another software design has a switch debounce time of 30ms [5]. This assumption could be used to explain the response times measured in these experiments. The response time to 'SET' (to initiate cruise control), shown in Figure 5.7, ranges from 28-76 ms. If the software design is consistent with that of [5], the switch debounce time could be approximated to be as much as 20 ms.

The error may also be due to the spare capability of the controller, which results in it being idle. In this case, the controller operates slower than estimated, since it is programmed to waste time on irrelevant tasks such as counting a delay loop.

5.1.3.6 Cruise Control Experiment Conclusions

From the measurement of the cruise control actual response time to switch inputs, the ECU was seen to take approximately twenty times the predicted response time, plus the probable switch debounce of approximately 20ms. The difference is believed to stem from the external and internal causes mentioned above. These can be summarised as follows:

- unknown software structure and the algorithm adopted by the designer
- microcontroller idle time to keep software timing structure
- allowance for mechanical component delays
- diagnostic procedures also being carried out by the processor

However, certain aspects of the cruise control operation seem to agree with the predicting model, as seen from the experiment. Those include:

- the average response time to the two inputs, SET and RES, are comparable, suggesting that the two are subject to similar control loops.
- the ECU monitors the state of the switches at one point in the loop. This is seen from the smaller deviation in response time when the output signals are used as switch inputs, to synchronise the measurement.
- vehicle speed has no effect on the response time of the ECU.

The above information learned from this experiment will be used in the next experiment and also in comparing electronic architecture requirements.

5.1.4 Air Suspension ECU Experiment

The second control system that was chosen for analysis was the air suspension system fitted to the Range Rover model year 1999.

This system is used to provide a soft and comfortable feel to a vehicle ride. Its function is to keep the four corners of the vehicle level, using air pressure. It consists of the following elements, air reservoir and air compressor which provide system air supply and maintain air pressure, four wheel height control valves, inlet and exhaust valves for wheel height control. Other than keeping 4 corners level, the ride height is also adjustable by the use of control buttons, UP and DOWN, to select between 4 different heights. The four heights in ascending order are 'ACCESS', 'LOW', 'STANDARD', and 'HIGH'.

During normal riding conditions, the four corner heights are constantly read from the sensors. The corner with the highest deviation from the required height according to the ride state, is corrected by means of opening that corner valve together with an inlet or exhaust valve to raise or lower down the corner, respectively.

The suspension system also responds to driver's inputs ('UP' and 'DOWN' buttons) and changes ride heights accordingly. When either a brake or door switch is on, the change between ride heights will be temporarily suspended.

Details of the software controlling the ECU were obtained from [1,7] and are described in the following section. The ECU contains a Motorola 8-bit MC68HC705B5 microcontroller, running at 4 MHz, as its central processor.

As in the previous experiment, the air suspension ECU response time to the switch inputs is predicted from its software specifications and compared to that actually measured experimentally.

5.1.4.1 Theoretical Response Time

5.1.4.1.1 Software Execution

From [1,7], the software cycle showed a simple timing structure, but did not employ a bespoke Real-Time Operating System (RTOS).

The software cycle is fixed at 10ms and is controlled by 20 interrupts, with a space of 0.5ms between them. The software is split into two main tasks: Synchronous and Sequential jobs.

5.1.4.1.1.1 Synchronous Job Modules

The synchronous jobs are to check the battery voltage, reset watchdog timer, control the triangular waveform supplied to the height sensors, and read the height sensor outputs. An interrupt is generated every 0.5ms, to start the synchronous tasks. Some of the jobs have no associated function ("Null job"), but exist to maintain the software timing structure.

As shown in Figure 5.16, each synchronous job is initiated after an interrupt every 0.5 ms. The synchronous jobs at time 0 and t ms, between 4.5 and 5.5 ms, of each software cycle are to control the timing of the triangular wave used to drive the height sensors. The synchronous jobs at time 3, 3.5, 4, 4.5, 8, 8.5, 9, 9.5 ms of each cycle read the height sensor inputs. Wheel height is taken as being proportional to the amplitude of the square wave sensor output. In order to minimise the effect of noise, each square wave height sensor output is read four times at high level and four times at low level. The sum of the four high values has subtracted from it the sum of the four low values, to average out the noise.

All of the synchronous jobs are limited to within 200 μs, inclusive of handling. For this reason, and also the fact that they are less computation, they are believed to take substantially less processor time than the sequential jobs.

The sequential job modules will now be described.

5.1.4.1.1.2 Sequential Job Modules

The sequential jobs perform the rest of the height control functions. As the name suggests, each task is executed after the previous one. They are executed during the time space between successive synchronous job interrupts. When an interrupt occurs, the sequential job is pre-empted. A scheduled synchronous job is executed and then the sequential job is continued after the completion of the synchronous job.

The software structure is as shown in the diagram of Figure 5.16.

The functions of the sequential jobs are described in brief as follows:

Service watchdog – reset watchdog timer

Calculate air spring height – divide height sensor reading by 4 (to complete the averaging of sensor reading), check validity of the readings and convert sensor readings into height.

Debounce vehicle status logic – debounce all the switches to accept state change after the new state has been stable for 30 ms. Check to inhibit simultaneous UP and DOWN switch press.

Process vehicle status logic – calculate the ride height required and control ride state change if demanded.

Check for sensor faults – check if the height sensor readings are within range, and set the appropriate fault flag and counter.

Settled at datum analysis – check if each wheel height settles into new ride height, after a change of ride state within time limit.

Calculate air spring logged errors – calculate the difference between actual wheel heights and demanded heights.

Increment excess error meters – check if any one corner height is consistently further away from demanded height than the other three.

Average logged errors – average wheel height of either front or rear axle depending on control mode.

Adjust air spring heights – use wheel height logged error and current height control mode data to determine the valve operations for wheel height control.

Drive outputs and select modes – check that there is no fault condition, before driving the outputs according to the demand determined in the previous job.

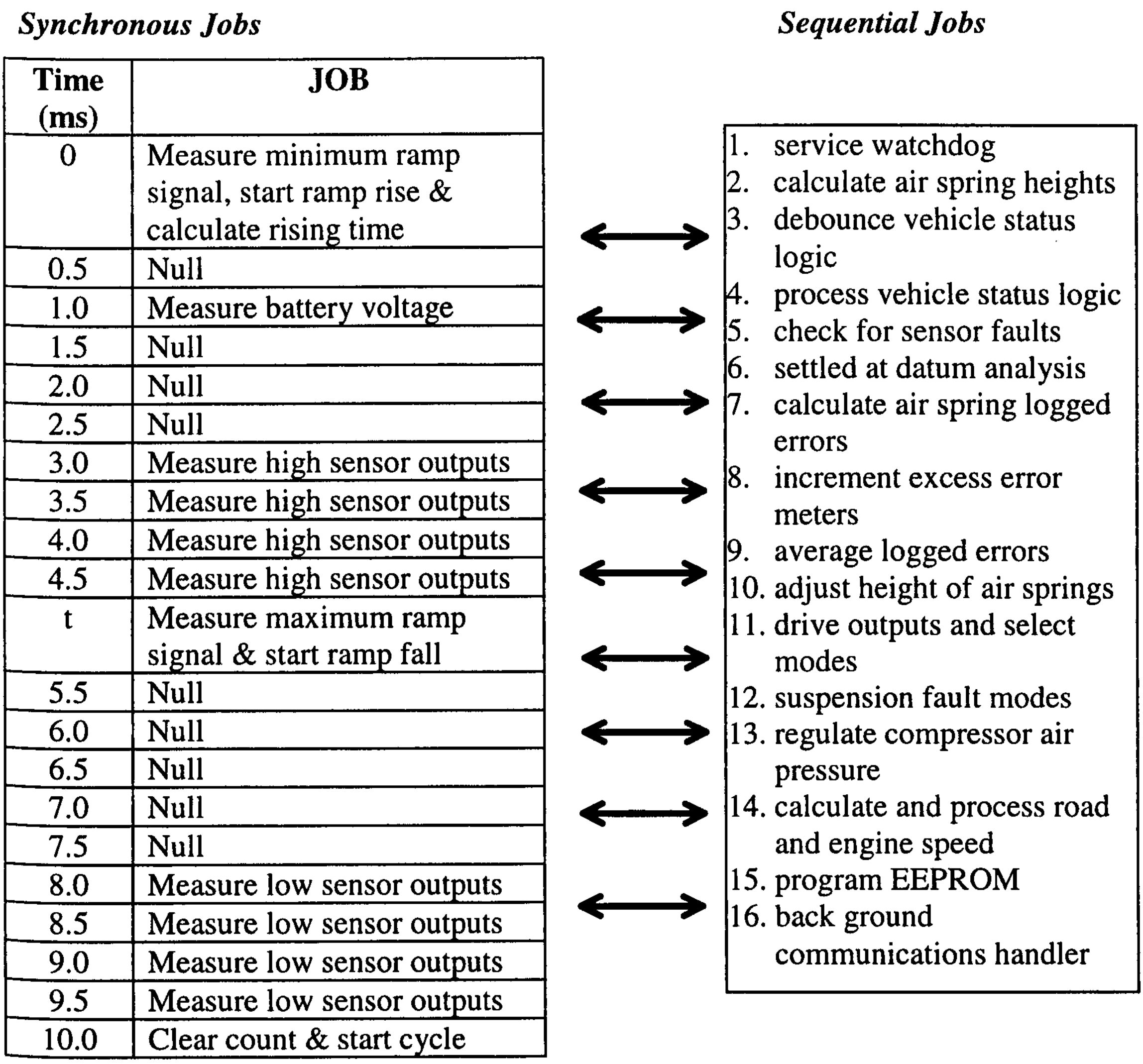
Suspension fault modes – analyse all the fault status recorded to determine if any of the valves is stuck.

Regulate compressor air pressure – monitor air compressor and diagnose for any fault

Calculate and process road and engine speeds – calculate road and engine speeds by counting their interrupt requests over the constant time period. The speeds are also range checked.

Program EEPROM – handle all the EEPROM reading and programming, as requested by other jobs. Check any change in fault register status and write the change on EEPROM.

Background communication handler – handle all the communication between ECU and the outside, such as diagnostic from laptop or sensor calibration.



Synchronous Jobs

Time (ms)	JOB
0	Measure minimum ramp signal, start ramp rise & calculate rising time
0.5	Null
1.0	Measure battery voltage
1.5	Null
2.0	Null
2.5	Null
3.0	Measure high sensor outputs
3.5	Measure high sensor outputs
4.0	Measure high sensor outputs
4.5	Measure high sensor outputs
t	Measure maximum ramp signal & start ramp fall
5.5	Null
6.0	Null
6.5	Null
7.0	Null
7.5	Null
8.0	Measure low sensor outputs
8.5	Measure low sensor outputs
9.0	Measure low sensor outputs
9.5	Measure low sensor outputs
10.0	Clear count & start cycle

Sequential Jobs

1. service watchdog
2. calculate air spring heights
3. debounce vehicle status logic
4. process vehicle status logic
5. check for sensor faults
6. settled at datum analysis
7. calculate air spring logged errors
8. increment excess error meters
9. average logged errors
10. adjust height of air springs
11. drive outputs and select modes
12. suspension fault modes
13. regulate compressor air pressure
14. calculate and process road and engine speed
15. program EEPROM
16. back ground communications handler

Figure 5.16 Air suspension software structure

The predictive model is to estimate the ECU response time to driver's control switches such as UP, DOWN and footbrake.

The detailed functions of the jobs are shown in the flowcharts together with modelled software pseudo codes and execution times in Appendix A.

5.1.4.1.2 Model prediction

Using the same method as on the cruise control system, a pseudo code version of the control tasks was written, and equivalent machine instructions devised to carry them out. The total machine instruction times were then calculated.

From the pseudo code shown in Appendix A, the response time to the 'UP' (change ride height) and foot brake switches (stop ride height changing) are calculated as between 31.2 and 41.2 ms. The response time is predicted as a range rather than an exact value, due to the fact that the software is structured in a loop. Switch debounce is set at 30ms. The switch status check is only performed once every 10ms of software cycle. If the switch is pressed just before the switch debounce module, the minimum debounce would complete in approximately 30ms. However, if the switch is pressed just after the switch debounce module just completed, it would take one complete cycle before it is starting to be debounced. Hence this maximum debounce time would be 40ms. The execution time taken by the modules between switch debounce and output control (when the response is detected) was calculated at 1.2 ms. Adding this estimation to the minimum and maximum debounce times, gives the response time prediction range of 31.2-41.2 ms.

The response to 'DOWN' switch was calculated at 1.031 - 1.041 s. The estimation method was the same as those responses to the other two switches, with a one second delay added as "direction_changed" check. This check is to ensure that vehicle is not changing vertical direction of travel within one second after the button pressed, before responding to the demand. This is believed to prevent grounding (in case the vehicle is travelling on bumpy road, it may ground when the suspension is lowered).

5.1.4.2 Practical Measurement of Response Time

5.1.4.2.1 Experiments

The experiments carried out were to evaluate the accuracy of performance prediction model by taking measurements of real-time ECU operations.

The ECU performance to be evaluated was the response to various switch inputs such as driver control 'UP', 'DOWN' and foot brake switches.

5.1.4.2.1.1 Circuit Connection

The air suspension circuit was set up for measurement. The connection is as shown in Figure 5.17.

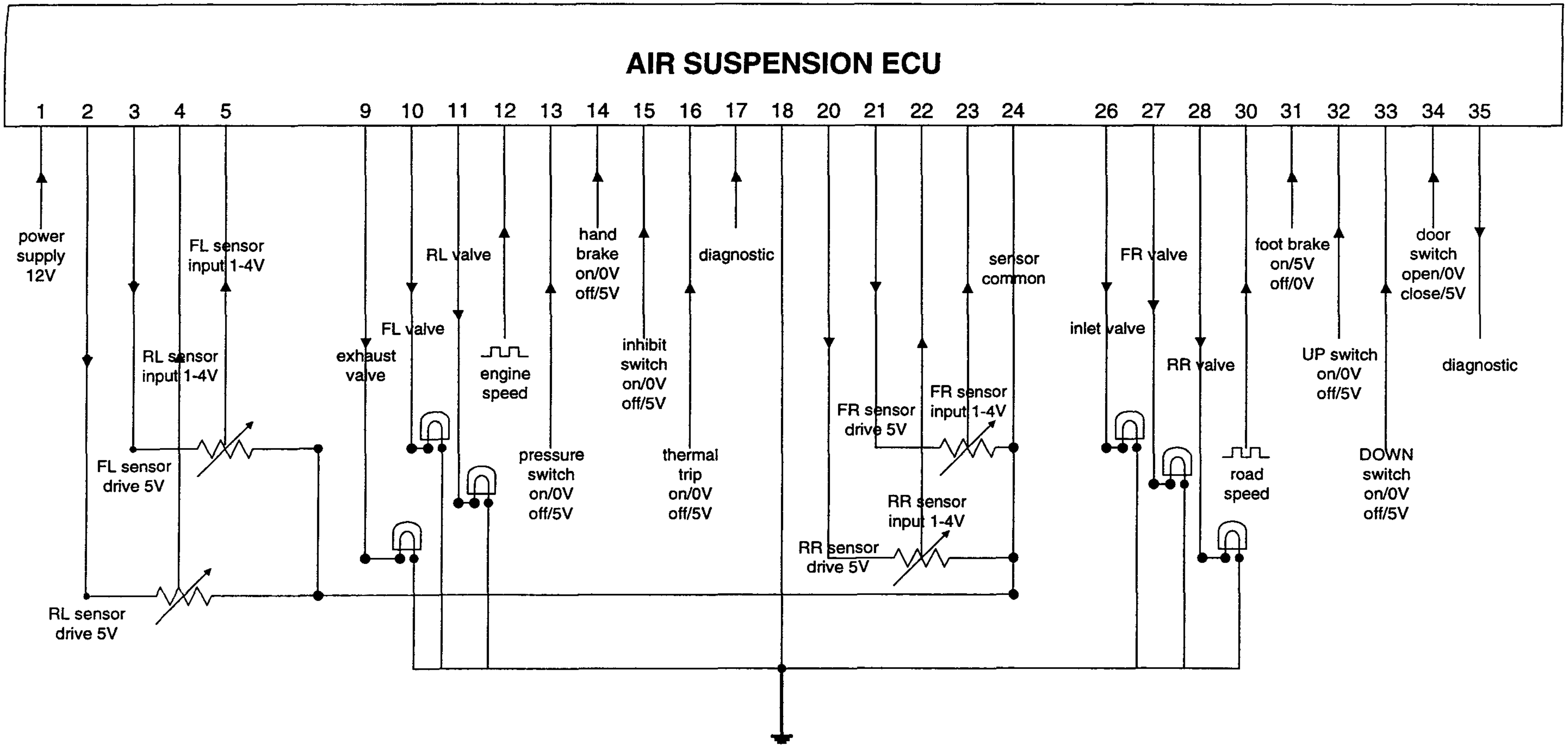


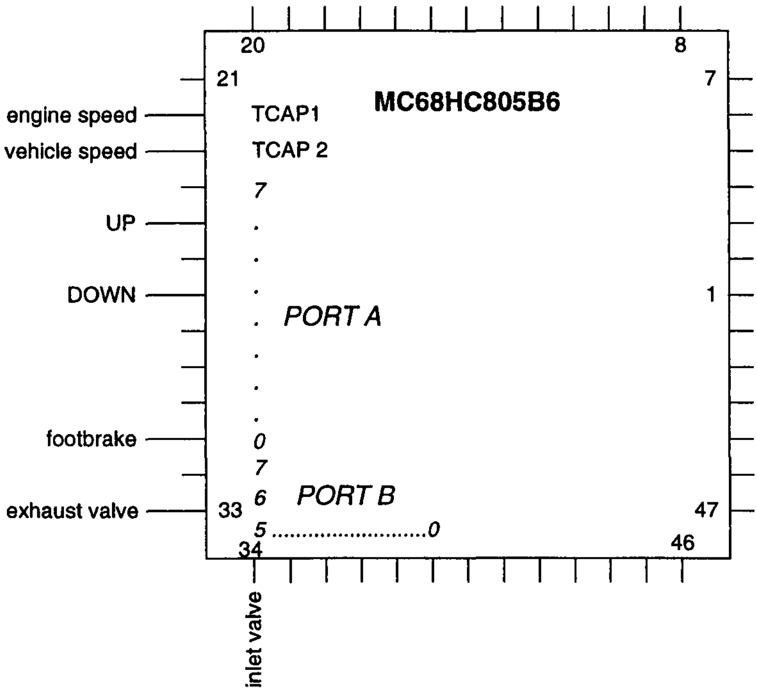
Figure 5.17 Air suspension measurement circuit connections

The external components used in this experiment were as follows:

- to observe the switching state of the air valves, they were represented by LEDs.
- handbrake, pressure, inhibit, and door switches were represented by toggle switches.
- UP, DOWN, and footbrake were controlled by button switches, connected to a 12 V supply.
- vehicle and engine speed signals were supplied by two signal generators.

5.1.4.2.1.2 Identification of Internal Connections

As carried out in the previous cruise control ECU measurement, the microcontroller connections of the air suspension ECU were inspected. The results were as shown in Figure 5.18.



TCAP - input-capture pin

Figure 5.18 Air suspension microcontroller connections

5.1.4.2.1.3 Measurements

Measurements were taken of the ECU response time to produce command signals to inlet and exhaust valves, initiated from the change in UP, DOWN or footbrake. When UP is pressed, the ECU would open the inlet valve to let air into the four corner valves, hence lifting the vehicle. On the other hand, the exhaust valve would be open to let air out, when DOWN is pushed.

The results on responses to 'UP' button and foot brake, to change ride state and inhibit change (during ride height change, if the foot brake is pressed, the process is halted), respectively, are shown in the graphs of Figure 5.19, 5.20. The response to 'DOWN' switch is shown in Figure 5.21.

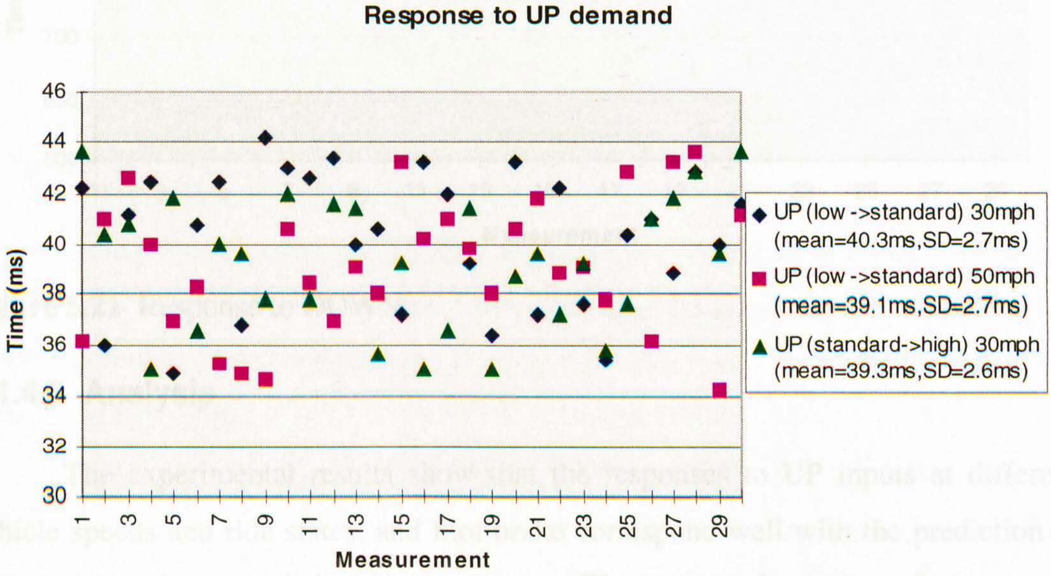


Figure 5.19 Response to UP demand

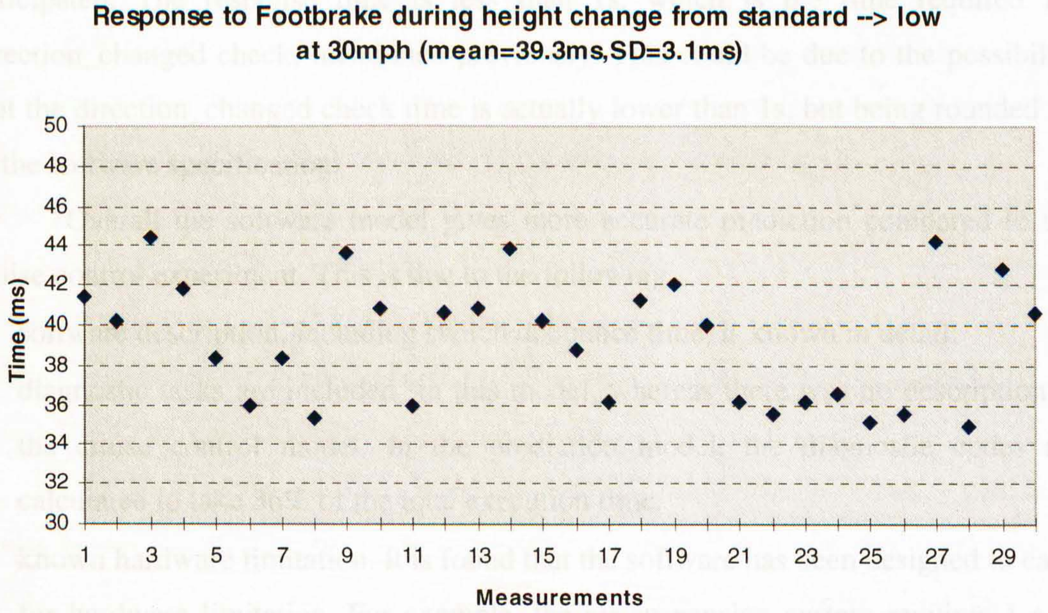


Figure 5.20 Response to footbrake during height state change

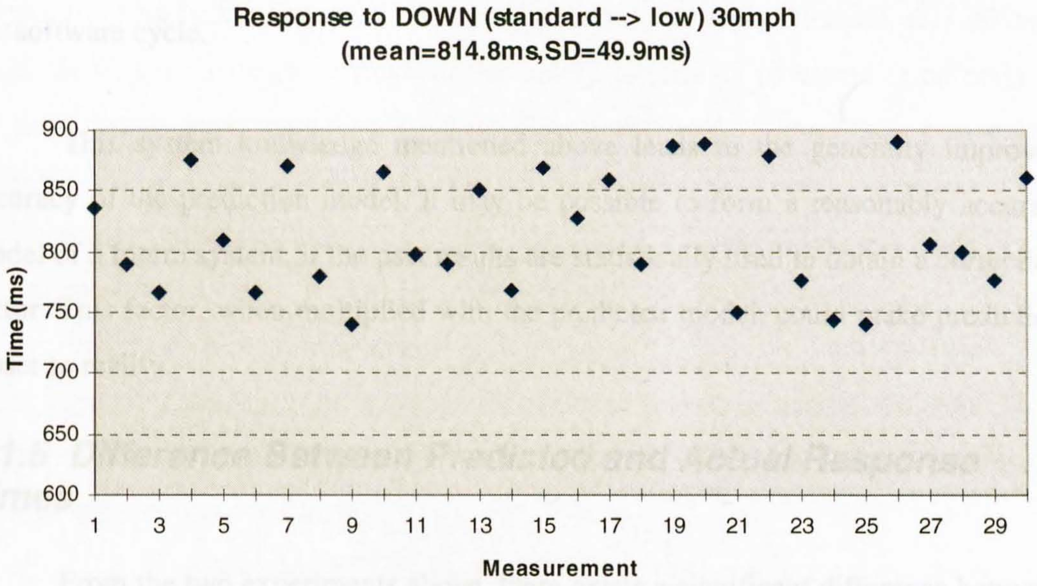


Figure 5.21 Response to DOWN

5.1.4.3 Analysis

The experimental results show that the responses to UP inputs at different vehicle speeds and ride states, and foot brake correspond well with the prediction in both mean and standard deviation measures. The measured response times ranged between 34.2-44.2 ms. The response to 'DOWN' button is, however, faster than anticipated. The response time is less than 1s, which is the time required for direction_changed check, mentioned previously. This could be due to the possibility that the direction_changed check time is actually lower than 1s, but being rounded up in the software specification.

Overall the software model gives more accurate prediction compared to the cruise control experiment. This is due to the following:

- software description, including switch debounce time, is known in detail.
- diagnostic tasks are included in this model, whereas there was no description in the cruise control model. In the prediction model, the diagnostic codes are calculated to take 36% of the total execution time.
- known hardware limitation. It is found that the software has been designed to cater for hardware limitation. For example, the air suspension system requires 1 s of drier time before any corner valve can be open. The software has to respond by having a delay before operating the valves. Hence it is believed that the

microcontroller generally has spare capacity, during which it stays idle, in a software cycle.

This system knowledge mentioned above leads to the generally improved accuracy of the prediction model. It may be possible to form a reasonably accurate model of a future system, if the past results are statistically used to obtain a correction factor. This factor, when multiplied with the predicted model, could make prediction closer to reality.

5.1.5 Difference Between Predicted and Actual Response Times

From the two experiments above, there exists a significant difference between the predicted and measured response times. The factors that were considered possible to attribute to this difference are:

- Diagnostics overhead

This is a feature of all safety-related software systems and typically occupies 30% of the total executable code. From the predictive model, the diagnostic code for the air suspension was approximately 36% of the total code. This would be the case for the future systems being proposed later.

- Switch debounce

This could be used for all user-inputs where mechanical controls were used. The experimental work by the author and the air suspension software information indicate there was a 20-30 ms overhead for each time the switch input changes.

- Real-Time Operating Systems (RTOS) overhead

Some complex electronic control ECUs, such as ABS and engine management system, employ RTOS. RTOS performs non application related tasks, such as scheduling and interrupt management, allowing the systems to be developed faster [18]. It, however, consumes microcontroller processing time and memory space.

This, however, was not apparent on the 2 relatively simple systems investigated, but would be likely to be used in more complex controllers, particularly if systems were combined in a single, complex controller. A large number of commercial RTOSs are available for most microcontrollers. RTOSs for each microcontroller vary in memory footprints and processor overheads. These characteristics are also application dependent, as RTOSs are highly customisable.

Software details of a particular application are needed for its RTOS overhead prediction. It is hence difficult to accurately predict its overhead at an early design stage. An evaluation of RTOSs of two 32-bit microcontrollers running an automotive powertrain control application, was published [26]. The results are that with 1000 interrupts per second, the RTOS overheads on CPU range between 13-30% for one microcontroller, and 15-33% on the other. With 2000 interrupts per second, the overheads on the two microcontrollers go up to 25-53% and 28-62%, respectively. It is suggested that RTOS speed and size are opposing optimisation goals, as an improvement in one tends to degrade the other [27]. The evaluation results from [26] also support this statement.

It is noted that RTOS overhead is linearly proportional to the number of interrupts, due to the interrupt intensive nature of powertrain control application, and interrupt latency being generally the most indicative factor in RTOS real-time performance consideration [28]. Context switching is the other factor often considered [29]. The powertrain control application also uses very small amount of other RTOS features.

The future systems will also have a large number of interrupts due to the high number of sensors and fast update rates. Its RTOS overhead is hence expected to depend on its number of interrupts. Furthermore, the impact of context switching on overhead can be minimised, by good software design that enables a processor to spends most time executing programs rather than switching between tasks [28].

- <u>Inefficiency due to compiler</u>

It is becoming unlikely that code will be written at the machine code level, due to large application size and complexity. Instead, compilers are regularly used to allow high level languages (C, C++) to be used with higher productivity by the programmer. These inevitably incur some inefficiency in their use of code. Literature source suggests that code size overhead generated by compilers could range from a factor of 1.2 up to 2.2 [30]. This same factor is believed to reflect on performance overhead also, since software performance is generally determined by code size [30]. However, the compiler efficiency could be improved significantly from changing its options, and versions to better suit the applications. This was demonstrated in [30] that the overheads of all of the four compilers studied could be improved to approximately 1.3.

The air suspension control was programmed using Assembly. The programming language of the cruise control is unknown. However, it is likely that the language was also Assembly, because of its similar complexity to the air suspension, both using 8-bit processors.

5.1.6 Correction (Multiplying) Factor

In order to predict the response time of other or unknown ECUs, the results gained from the two experiments on cruise control and air suspension ECUs will be used to derive a correction factor.

Taking the above influencing aspects into account, the correction factor should be calculated by first taking the maximum switch debounce time off the measured response time. Then the predicted response time, added by a 36% of approximated diagnostic codes (assumed equal to that of the air suspension), is divided into the measured response time. The result is the correction factor. As discussed earlier, both systems are believed to be without RTOS and written in Assembly language, so the effect of RTOS and compiler are not included here.

5.1.6.1 Cruise Control Experiment

The largest response time to 'SET' and 'RES' (resume) in the cruise control experiment, shown in Figures 5.7 and 5.13, is 76 ms. The predicted response time is 2.65 ms. Adding the diagnostic code estimate gives 4.14 ms. The design switch debounce time is estimated at 20 ms, from the range of the above results between 23-76 ms, and the air suspension switch debounce time of 30ms. Since both the air suspension and cruise control ECUs have simple 8-bit central microcontrollers, their level of software complexity should be similar. The software loop of the cruise control is hence estimated at 10 ms, the same as the air suspension. As described in the previous section, the largest possible debounce time is approximately the sum of the software loop and the design debounce time. This makes the cruise control largest response time excluding switch debounce to be 46 ms. The cruise control multiplying factor is, therefore, 46/4.14 = *11.1*.

5.1.6.2 Air Suspension Experiment

The largest response time to the 'UP' switch of the air suspension without switch debounce time is 4.2ms, where as the predicted value stands at 1.2ms. The correction factor is therefore 3.5. As mentioned earlier, the difference between the predicted and measured response times of the air suspension ECU are much smaller than that of the cruise control, due to having much more software information available. However, there is no such information on the software of other systems to be predicted. The only available information will be system operations, from user manuals or articles, as in the cruise control case. This ratio is, therefore, unsuitable to be used as a multiplying factor.

In order to establish a multiplying factor from the air suspension experiment, the software prediction model is to be reviewed. By not taking into account of the software details and only using vehicle user manual [1], the software model would only include sections ❶, ❸, ❺, ❻, ❾ (sub section 10), and a quarter of A/D conversion from synchronous jobs (taking only one reading instead of four from each wheel height sensor). The predicted response time would be 0.39ms and 0.61ms with diagnostic estimate added. The multiplying factor would be 4.2/0.61 = *6.9*.

The average of the multiplying factors from the cruise control and air suspension experiments is *9*. It will be used in other ECU response time prediction.

5.1.7 Conclusions from the Two Experiments

The two experiments were executed in an attempt to use software modelling to predict the response time of the cruise control and air suspension system ECUs. The response time predicted by the models were compared to the measurement results on the actual ECUs. The measured and predicted response times are clearly different. The influencing factors to the difference, including switch debounce, diagnostic and RTOS overheads, and compiler inefficiency, are discussed.

The predictions on the cruise control ECU was also less accurate than those of the air suspension ECU. This was due to the fact that there was more software information on the air suspension system than on the cruise control. Software structure, switch debounce time, as well as part of the control algorithm and diagnostic information was available on the air suspension, whereas the cruise control flowchart was only derived from a vehicle user manual.

Based on the influencing factors and different level of software information above, the correction factors from the two experiments were derived. The average correction factor of *9* will now used in other ECU response time prediction.

5.2 ECU Performance Requirement of Different Architectures

The software modelling technique from the above experiments will now be used to predict response time of other control ECUs of different electronic architectures. This model, together with memory estimation and the wiring model (from chapter 6) will be used to estimate the specifications of microcontrollers, required to fulfil the response time limits.

5.2.1 Microcontroller Selection Criteria

There are many factors that system developers need to take into consideration, in selecting a microcontroller for their applications [8-12]. Those criteria include hardware peripherals e.g. ROM, RAM and I/O, processing power, price, availability, development tool availability, manufacturer support, compatibility with earlier systems, etc.

This project is intended to do a feasibility study of different vehicle electronic architectures. The actual system development is beyond its scope. The focus is, therefore, on specifying what specifications a microcontroller needs to possess to perform the control tasks. Hence, only hardware peripheral and processing power requirement will be examined. The overall system costs will also be estimated and discussed in Chapter 6.

5.2.2 Alternative Architecture ECU Response Time Modelling

The exemplary ABS, 4WS and active suspension systems, described in Chapter 3, are studied and their software control flowcharts and pseudo code are derived. They are shown in Appendix A.

5.2.2.1 Microcontroller Family

As the complexity of the various control tasks was not known at the outset, the target microcontroller could not be sensibly selected. Instead, a range of microcontrollers was chosen, with the most suitable becoming clear from the software modelling exercise.

The instruction set used in the model belongs to microcontrollers from Motorola. Motorola microcontrollers were selected due to a number of reasons. Firstly it is due to familiarity since the air suspension previously experimented contains an 8 bit Motorola microcontroller, so its instruction set was already studied. Secondly their microcontrollers are widely used, and there are a large number of textbooks based on their products [9,12]. Motorola also produces microcontrollers ranging from 8, 16 to 32 bits, with a complete online documentation.

The M68HC05, M68HC08 and M68HC12 were initially selected for software modelling. M68HC05 is the most basic 8 bit Motorola microcontroller, while M68HC08 is more powerful, and is software compatible with the former family. M68HC12 is one of the two 16 bit Motorola microcontroller families.

5.2.2.2 Response Time Estimation

This section describes the transformation of modelled system pseudo codes into response time estimation.

The control execution time from the pseudo codes is obtained from equation 5.1. The multiplying factor of 9, derived from the two experiments, is multiplied to the control execution time, to obtain the total execution time. The effects of the influencing factors, which could cause the difference between the predicted and actual response time, are then taken into account. They are multiplied to the total execution time. The calculation is done as shown in equation 5.3.

$$T_E = T \text{ x } C \text{ x } O_{diag} \text{ x } O_{cpl} + O_{RTOS} \quad (5.3)$$

T_E - estimated response time
T - control execution time
C - correction (multiplying) factor
O_{diag} - percentage of diagnostics overhead
O_{cpl} - percentage of compiler overhead
O_{RTOS} - RTOS overhead

Diagnostics overhead of 36% of the total code (equal to 56% of control code) from the air suspension experiment, which is within a range of typical diagnostic percentage of up to 40%, is used here [20].

Independent valuation study on compiler overhead for individual microcontrollers considered here is not available. Typical compiler overhead can range up to a factor of 2, and the worst case of up to 3 [21]. However, [30] demonstrates that after several revisions, the factor could be improved down to approximately 1.3, for all the compilers studied. With the heavy financial constrain of the automotive industry, it could be assumed that software developers would devote to the same practice, to reduce ROM requirement and hence cost. The same factor is, therefore, used for the compiler overhead study here.

Commercial RTOSs come in a wide range of performances for a single application. To gain an accurate RTOS overhead, detailed software operation is needed, and with corroboration from a vendor, software developers can customise an RTOS to suit their applications in terms of performances and memory usage. In this stage of design, an approximation of RTOS overhead can only be made from their performance study of similar applications.

RTOS overhead is taken from the only available evaluation study of ten RTOSs of a more powerful Motorola MC68332 microcontroller, running powertrain control system in [26]. The automotive powertrain application involves a lot of interrupts. This is, to smaller extent, also true for dynamic control systems, due to their many sensors and high speed update rates. The overhead of a medium performance RTOS is used, as it has a compromised performance between overhead and RAM usage. The overheads are proportional to the rate of interrupts. The overhead plot of the selected RTOS is shown in Figure 5.22.

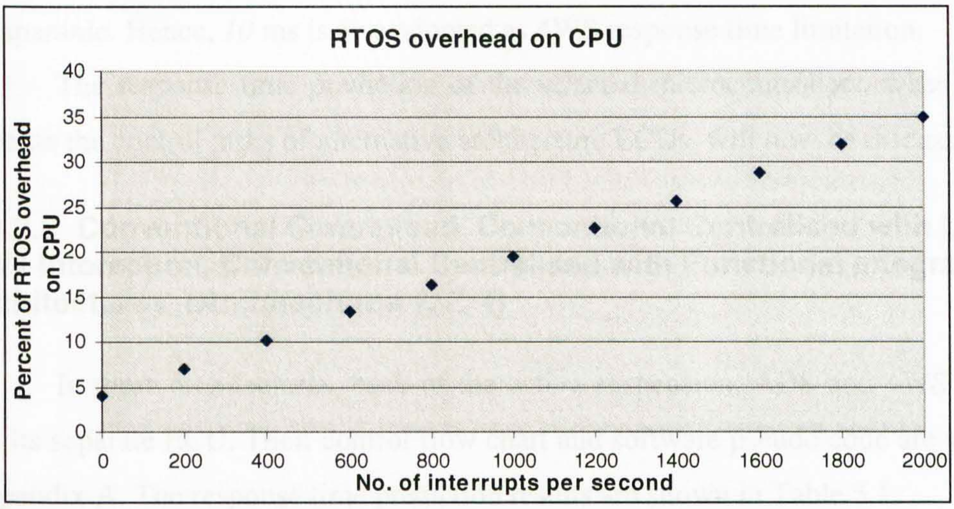


Figure 5.22 Percent of RTOS overhead on CPU

The number of interrupts per second of the predicted systems can be estimated from the number of sensor inputs, and their update rate. However, the RTOS overhead percentage from the evaluation was calculated based on only powertrain application. An actual RTOS overhead for a specific number of interrupts should be almost constant regardless of applications. If a less time consuming main task than a powertrain control had been used in the evaluation, the percentage of RTOS overhead would have been higher. To maintain the consistency for comparison of all the systems being predicted, a virtual application execution time of 5ms was assumed. This was used as a main task execution time for RTOS overhead calculation, for all of the predicted systems. The value 5ms was selected because when added with RTOS overhead, the total system execution time should be within the 10ms limit (of the predicted systems here) with some spare capacity. This would be a realistic timing target for a system developer.

A microcontroller is considered capable of performing the ABS, active suspension or 4WS control tasks if its response time is within the specified limits. The exemplary active suspension system calculation is executed in cycles of several milliseconds [13], while the air suspension in the experiment has a 10ms cycle. *10* ms is, therefore, adopted as the active suspension response time limitation. Two of the ABS systems have software cycle times of 5 and 6 ms [14,15]. The more stringent *5* ms is selected for ABS response time limitation. None of the 4WS systems in the literature explicitly states its software cycle or response time, with one system reported having its control carried out at high speed [16]. Since the active suspension and 4WS have been integrated experimentally [17], their response times are assumed comparable. Hence, *10* ms is also adopted as 4WS response time limitation.

The response time prediction of the selected microcontrollers, when used to execute the control tasks of alternative architecture ECUs, will now be discussed.

5.2.2.3 Conventional Centralised, Conventional Centralised with Limited CAN Interaction, Conventional Centralised with Functional Integration Architectures (*Architectures 1, 2, 4*)

In these architectures, each of the active suspension, ABS and 4WS systems has its separate ECU. Their control flow chart and software pseudo code are shown in Appendix A. The response time prediction results are shown in Table 5.1.

Architectures 1, 2, 4	**Active Suspension**		**4WS**		**ABS**	
	Total response time (ms)	Response time limit (ms)	Total response time (ms)	Response time limit (ms)	Total response time (ms)	Response time limit (ms)
M68HC05 (8 bit)	7.9	10	**3.5**	10	21.7	5
M68HC08 (8 bit)	**2.7**	10	1.1	10	**4.7**	5
M68HC12 (16 bit)	2.3	10	1.0	10	4.5	5

Table 5.1 Response time prediction results of *Architectures 1, 2 and 4*

The results suggest that all three ECUs could be controlled by 8 bit microcontrollers. It should be noted that *Architecture 4* would realistically take longer response time to control than suggested by the model, since each system would have to execute extra combined control tasks. Since the details of integrated control is not available and various system developers have different approaches, the tasks were not included in the software models.

5.2.2.4 Total Centralised Architecture (*Architecture 3*)

The only ECU of this system performs the all of active suspension, 4WS and ABS control tasks, requiring their total execution cycles (T_c) to be added together. With the response time limit of 5 ms, the ABS control has to be executed twice under the system response time limit of 10ms. Hence the ABS total execution cycle is doubled before adding to those of the 4WS and active suspension, whose limit is 10 ms. Its response time prediction results are shown in Table 5.2. The details are in Appendix A.

Architectures 3	Total response time (ms)	Response time limit (ms)
M68HC05 (8 bit)	51	10
M68HC08 (8 bit)	9.5	10
M68HC12 (16 bit)	**8.6**	10

Table 5.2 Response time prediction results of *Architecture 3*

The result suggests that the M68HC12 microcontroller is capable of performing the *Architecture 3* control tasks. Both this and the 8 bit M68HC08 microcontroller have comparable response times for this task.

5.2.2.5 Distributed Wheel Controller Architecture (*Architecture 5*)

This architecture consists of one central controller and four distributed wheel controllers. The 4WS tasks are solely controlled by the central ECU, while those of ABS and active suspension control are split between the ECUs. The ABS and active suspension control of a wheel is carried out by the corresponding distributed wheel control, while the central ECU computes shared data such as vehicle speed. From the ABS and active suspension flow charts in Appendix A, the ABS control task ❷ and active suspension control task ❶ are executed by the central ECU. The rest of the control of is done by the distributed wheel ECUs.

The software modelling results are displayed in Table 5.3.

Architecture 5	**Central ECU**	**Distributed ECU**	
	Total response time (ms)	Total response time (ms)	Response time limit (ms)
M68HC05 (8 bit)	**4.5**	17.8	10
M68HC08 (8 bit)	1.4	**2.2**	10
M68HC12 (16 bit)	1.2	2.1	10

Table 5.3 Response time prediction results of *Architecture 5*

The results suggest that an M68HC05 microcontroller is capable of executing the central ECU control tasks within limits, while the more complex M68HC08 microcontrollers are needed for each of the four distributed ECUs.

5.2.2.6 Analysis

The software modelling shows that all of the control, of the ECUs of the five architectures, can be done by 8 and 16 bit microcontrollers. However, the software model does assume that the resolution of control variables in all systems is within 8 bits. This information on this resolution is not available in literature. Some of the existing systems may have higher resolution, which would require longer execution time from 8 bit microcontrollers. To do 16 bit number calculation such as addition an 8 bit microcontroller has to carry out lower byte and then higher byte additions. The process would require approximately twice the number of execution cycles and hence twice the execution time as an 8 bit number addition. 16 bit number multiplication and division would take more than double of that execution time.

Hence, considering the predicted ECU response time of all the architectures, only the 8 bit microcontrollers with at least 50% spare capacity (response time limits are much more than twice as much as the predicted response times) were selected.

The results suggest that, in terms of processing power, it is feasible for all of the architectures to be implemented using current microcontrollers available in the market.

5.2.3 Active Suspension, 4WS, and ABS Memory and I/O Predictions

Having predicted the response times taken by different microcontrollers in the five architecture control tasks, now their memory space and I/O required will be predicted.

5.2.3.1 ROM Estimation

Memory estimation, as in response time estimation, is generally obtained from experience from earlier developments of similar systems. The only ROM estimation method, which requires no detailed software information, found in literature [22] is based on the statistical technique [31]. The work was developed by computer scientists to roughly estimate the length of high-level language programs (number of program lines) from preliminary design information. From the program length, [22] gives an approximation of 5-20 bytes of machine code for each line, and hence the overall ROM estimate. For worst case estimate, the number of machine code for each line of 20 was used here.

The estimate is based on the number of operators and operands in a program. Operators are program commands that act on variables or control program flow, such as mathematical signs, for examples +, -, = and >, or commands such as *FOR*, *IF-ELSE*, *GO TO*. Operands are elements of a program that are acted on by operators, including variables (external and internal), constants, function or subroutine names.

The program length estimate is governed by equation 5.4 [31].

$$L = N/n, \quad N = k(0.5772 + \ln k) \qquad (5.4)$$

L - number of program lines
N - estimate of total number of operators and operands
n - a typical number of operators and operands in each program line
k - sum of total number of distinct operators in a programming language and estimate number of distinct operands in the program

The total number of distinct operators in C, the language assumed to be used here, is 53 [32]. The estimate number of distinct operands is the sum of the number of external (I/O) and internal variables, constants, and processes. A typical number of operators and operands in each program line, n, is between 3 and 5. For example:

A = B has one operator and two operands.

A = B + C has two operators and three operands.

For an upper approximate, n=3 is used here. As practised in [22], 50% extra number of operands is allowed since it is not possible to identify all of them at this stage.

It is acknowledged that this estimation method is approximate, being given an accuracy range of approximately up to 45% [22]. The author also believes it to be conservative. It is likely that the programs, which were studied and used to derive the formula, do not contain as much amount of diagnostic check on each data as highly safety related automotive control software. This may cause an underestimation of program length. However, as this technique is used consistently on all the predictions, it should provide a useful memory comparison between systems and a guideline for microcontroller requirements. RTOS memory footprints are not included in the estimate, due to the fact that there is a wide range of ROM usage between commercial RTOSes. A minimum ROM footprints of commercial RTOSes for M68HC12 range between 2.5 and 14 kbytes [19]. Since RTOS kernel can be customised to suit an application which would inevitably changes its memory size, it would be unrealistic to estimate RTOS ROM space at this early stage.

The calculation details are shown in Appendix A.

5.2.3.2 RAM and I/O Predictions

RAM requirement could be estimated from program variables [12,22]. However, there are many unknown variables in the program other than the known ones from the control algorithms. Furthermore, the multiplying factor, derived from the response time experiments, can not be assumed to indicate the ratio of variables from the flowcharts and total variables in the program. For these reasons, RAM space required is instead estimated from ROM space. [7] suggests that RAM could be estimated to be 12 and 20 times less than ROM space, for assembly and high-level language programming, respectively. Since it is assumed that the ECU software is programmed in high-level language, RAM to ROM ratio of 1/20 is used. This estimate is again believed to be conservative, since it relies on the ROM estimate described above. It is suggested by [22] that it is extremely important that RAM should be generously estimated at this preliminary stage, since the software development cost will spiral up if a program needs to work with an insufficient memory space. RAM estimate is hence added 50% more space for unforeseen variables in later design stage.

The number of I/Os needed is simply taken from the wiring models. This should provide reliable estimate, since the wiring models are formed using signal information of existing systems.

5.2.3.3 Memory and I/O Prediction Results

Following the prediction methods described above, the memory and I/O predictions of the ECUs of the five architectures are as shown in Table 5.4.

	ROM (k bytes)	RAM (k bytes)	I/O
Architecture 1			
Active suspension ECU	7.6	0.6	28
4WS ECU	5.0	0.4	7
ABS ECU	7.1	0.5	38
Architectures 2, 4			
Active suspension ECU	7.6	0.6	27
4WS ECU	5.0	0.4	7
ABS ECU	7.1	0.5	30
Architecture 3			
Total centralised ECU	17.2	1.3	63
Architecture 5			
Central ECU	10.4	0.8	42
Distributed ECU	7.8	0.6	6

Table 5.4 ROM, RAM and I/O requirement Predictions of all the architectures

It should be noted that RTOS RAM usage was not included in the prediction, for the same reason as the exclusion of RTOS in ROM estimation. However, the evaluation study of commercial RTOSs in [26], described earlier, suggests that using standard commercial RTOSs for an M68300 32-bit for automotive powertrain application is impractical. This is because RTOSs consume too much on-chip RAM. This argument is also supported by [33]. This could also be the case for the microcontrollers in this study. Most of the M68HC08 and M68HC12 contain 1Kbytes of RAM, whereas maximum RAM for M68HC05 is 0.5Kbytes [34]. Considering a 4WS ECU, an M68HC05 could perform its control task but with 80% of its RAM capacity used. It is very likely that if the M68HC05 is to be used for 4WS ECU, it has to be with in-house RTOS or without it at all. This unsuitability of conventional RTOSs for automotive control applications is one of the reasons that European OSEK (Open systems and the corresponding interfaces for automotive electronics) committee produced its OS definitions. Other benefits of OSEK OS definitions are the portability of ECU applications from different suppliers and interoperability of ECUs in a vehicle network [23]. The definitions allow for small RTOS kernels demanding small ROM and RAM compared to conventional RTOSs. Although there are some

OSEK RTOSs in the market [35,36], they are not yet widely used since they are new to the market. Besides, OSEK OS definition was just developed and it may still have some design flaws [34].

The ROM, RAM and I/O requirement results also confirm that all of the electronic architectures considered, in term of microcontroller specifications, could be implemented using current technology.

5.3 Controller Area Network (CAN) Simulation

CAN plays an important part in providing a means for high speed data transfer between all the ECUs in the vehicle considered in this thesis. The powertrain and dynamic control ECUs are safety related, so it is vital that the data transfer between these ECUs occurs within allowable time limits for the control tasks. This section, therefore, concentrates on establishing that correct operation of the specified architectures can be achieved within these specified limits. It covers the relevant characteristics of the CAN protocol, the chosen simulation method, the simulation input data and the analysis method to be used.

5.3.1 Timing Aspects of CAN

When a CAN bus is free of messages, any node is able to send an information on to the bus. However, in the event of two nodes trying simultaneously to transmit data, a method is needed to avoid data corruption and excessive loading on the bus. To do this, CAN employs bit arbitration process in determining the right to transmit a message, when more than one node tries to send their data. The nodes which send simultaneous messages continue their message transmission, while keep comparing their own message with the bus status it by bit. Simultaneous transmission can continue for as long as all the bits of the signals sent are the same. Once a bit difference is detected (indicating a collision), the message (or messages) which contains a passive bit (1) will be stopped, while the one with a dominant bit (0) will continue. The node(s) that backs off the transmission will try to send its signal again as soon as the CAN bus is free.

By giving each data a unique identification number, same priority message collision where all the messages have to be taken off line, can be avoided. In practice, more important or urgent messages are given lower identification numbers, which give them higher priorities to be transmitted. Under the CAN protocol, the message with the highest priority always get the first access to the bus, which effectively guarantees its delivery time. It can be seen that a problem can arise when a large number of high priority messages are transmitted, and lower priority messages will have to keep backing off transmission. This may cause long delays before lower priority messages can be sent.

Four out of the five vehicle electronic architectures considered in this thesis could potentially employ CAN. It is, therefore, important that the CAN message delays of these architectures are examined. An excessive delay of signal transmission is a delay longer than the period of the transmission, causing the delayed data to be obsolete. This is shown in Figure 5.23. The delay would probably force the relevant ECU to employ previous data, not up to date, in its control tasks instead. This could cause inaccurate control. The effect of the delay may not be significant in non safety related data such as climate control but it is vital for most of safety related class C signals such as wheel speed sensors. Hence an excessive delay of these class C signals in any of the architectures may render it unfeasible or unsafe to implement.

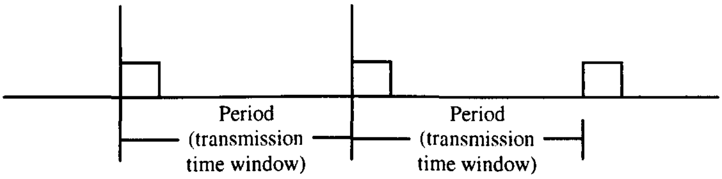


Figure 5.23 A series of periodic signal getting transmitted

Thus, a simulation of CAN data transfer between ECUs has been carried out. In order to realistically simulate this operation, real information on data signalling in the target vehicle is needed, including all the other messages that would be using the CAN bus. The work in [15], which also studies the timing characteristics of CAN, provides this information. The information contains common messages shared between ECUs of a modern luxury car [24]. As stated in [15], this information represents a generic workload, that can be used to characterise background network traffic of a vehicle, and will now be described.

5.3.2 Simulation Data

The information comprises 90 periodic signals that are shared between 9 CAN nodes, namely the Powertrain Control Module (PCM), Anti-Lock Braking System (SRS), Passenger Safety System (PSS), Air Suspension/EVO Steering (ASE), Ignition Control Module (ICM), Instrument Cluster Display (ICD), Trip Computer Diagnostics, Cruise Control System (CCS), Climate Control Module (CCM). The signals include all classes (Class A, B and C), characterised as low, medium and high speed messages, respectively.

From the data, class C signals have transmission frequencies of 200 and 100 Hz. Those frequencies of class B signals are 50, 10 and 5 Hz. Class A signals are transmitted at 1 and 0.1 Hz. The data details and how it is prepared is described in the following section.

5.3.2.1 Data Preparation

From the list of the 9 CAN nodes above, there is a difference between the electronic systems specified in the supplied data, and those that have previously been considered in this thesis. Notably, PSS and SRS systems here cover the function of the ABS/ASR system modelled in the thesis. Furthermore, ASE system in the simulation data combines the equivalent functions of active suspension and 4WS systems, modelled in this thesis.

Since the ABS/ASR, active suspension and 4WS are focused upon in this thesis, they are to be treated as individual nodes. Therefore, the ASE and its signals are removed, and replaced by the active suspension and 4WS systems and their signals from the exemplary systems in Chapter 4. Similarly, the exemplary ABS/ASR system and all of its control signals are introduced to the simulated vehicle, in place of SRS system. Also all the brake control related signals are taken away from the PSS node, while it retains the rest of its signals, which are used for airbag and other safety controls.

The signal list contains all the sensor and actuator signals, some of which are not transmitted on the CAN bus in the wiring diagrams in Chapter 6. However, it is intended to keep the number of signals on the CAN bus high, so that the CAN bus can be simulated under high workload. This is to examine the systems' behaviour under worst case conditions and also because more electronic systems and consequently more signals are going to be transmitted on the CAN bus in the future. By keeping the number of signals high, the simulated vehicle will more closely resemble vehicles in the near future.

Due to the above reason, a sensor node is introduced into the system. It transmits ABS/ASR, active suspension and 4WS sensor data onto the CAN bus. For the same reason, the ABS/ASR, active suspension and 4WS control signals are assumed to be transmitted onto the CAN bus.

The transmission frequencies of the newly introduced signals are kept consistent with the rest of the list. For example, the ABS/ASR wheel brake controls are given the same frequency as the wheel speed sensors, and new warning lamp signals are given the same frequency as the ones on the list.

The complete list of 117 signals of this CAN simulation data is shown in Appendix B.

The number and the combined bandwidths of the signals of different frequencies are shown in Figure 5.24 and 5.25.

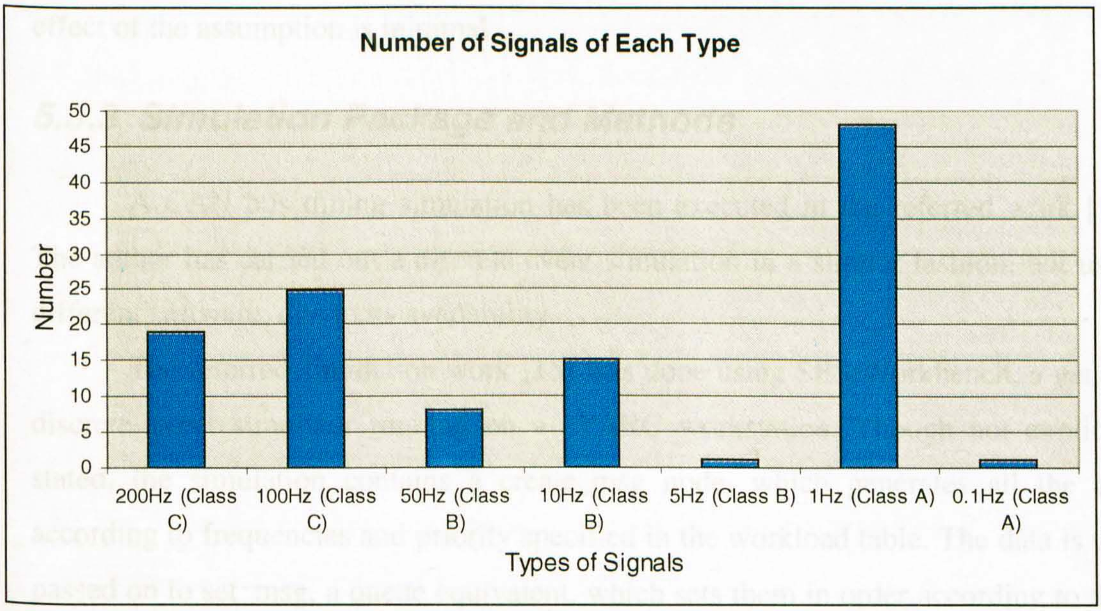


Figure 5.24 Number of signals of each type

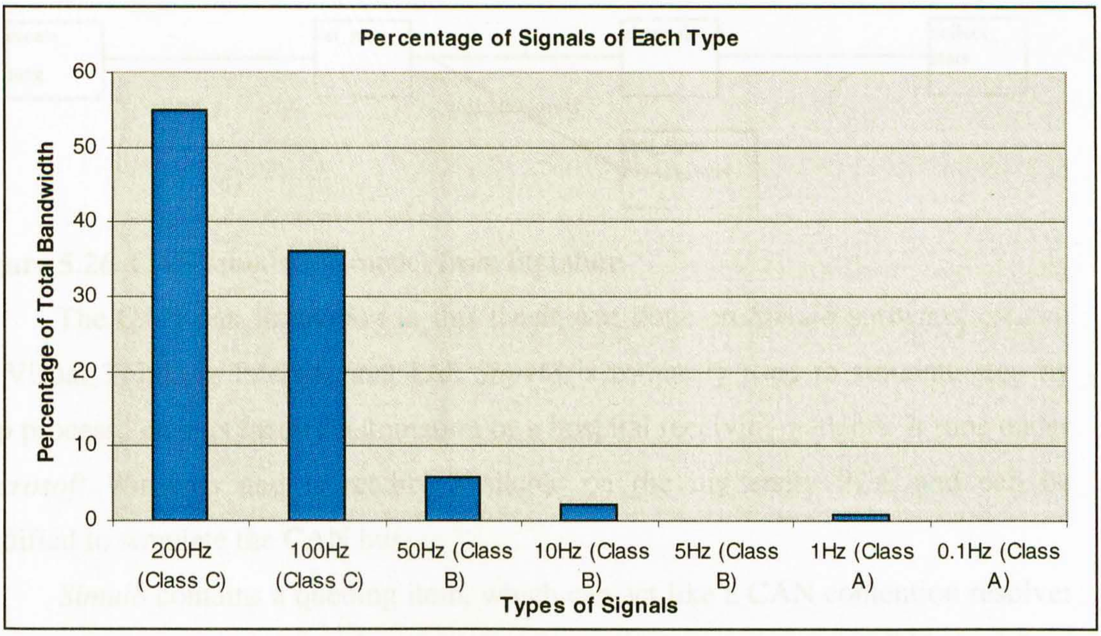


Figure 5.25 Percentage of bandwidth of different types of signals per total bandwidth

Note that the percentage of class A signal bandwidth to the overall bandwidth is very small as seen from Figure 5.25. Due to this fact, some of the class A signals such as horn or L/R indicator signals controlled by the driver, whose nature is not periodic, are assumed periodic for simplicity. Since their bandwidth is very small, the effect of the assumption is minimal.

5.3.3 Simulation Package and Methods

A CAN bus timing simulation has been executed in the referred work [15]. The author has carried out a discrete event simulation in a similar fashion, but using different software, due to its availability.

The referred simulation work [15] was done using SES/Workbench, a generic discrete event simulator running on a SPARC workstation. Though not explicitly stated, the simulation contains a create_msg node, which generates all the data according to frequencies and priority specified in the workload table. The data is then passed on to set_msg, a queue equivalent, which sets them in order according to their priority. From there, they go onto the channel, the CAN bus equivalent. Timing statistics of all the messages are collected. The simple model of the simulation is shown in Figure 5.26.

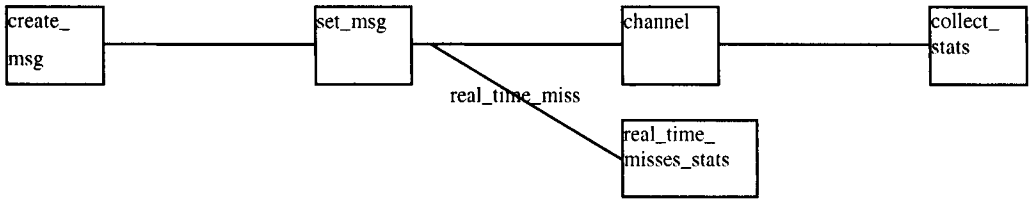


Figure 5.26 CAN simulation model from literature

The CAN bus simulation in this thesis was done on *Simul8* software, created by Visual Thinking International Ltd. *Simul8* is primarily used to simulate step by step process, such as factory automation or a hospital receiving patients. It runs under *Microsoft Windows* and is readily available on the university PCs, and can be modified to simulate the CAN bus.

Simul8 contains a queuing item, which can act like a CAN contention resolver by letting the highest priority one among the waiting messages go on CAN bus first. As soon as the CAN bus is free, the next highest priority message is allowed to be sent.

Furthermore, *Simul8* provides a source item, which can be used to represent a CAN node. Its function is to arrange its messages and attempt to send the highest priority one first when more than one messages are ready to be transmitted. From these functions, the timing aspect of CAN bus access can be simulated accurately.

5.3.3.1 Assumptions and Data Settings

In this CAN simulation the following assumptions are made:

- all the signals are assumed to be short (2 bytes) and periodic which is the nature of real-time data such as sensor and control signals. Each message, therefore, is eight byte long, containing the two bytes of data and six bytes of CAN overhead as in [23].
- all the messages are given priorities according to their transmission frequencies. Ones with higher frequencies have higher priorities.
- messages with the same transmission frequencies are given the same priority. There are 7 different frequencies and hence 7 priority levels. This is not exactly like actual CAN application, in which each message has its unique priority level. The messages were not given unique priority levels because it was preferred to study the transmission delay of a group of messages with the same frequency,

rather than individual messages. Besides, this practice does not affect the timing of CAN bus access. Same priority level messages would gain access to the CAN bus arbitrarily. It could be assumed that the one that misses out has lower priority. In terms of timing, one message gets access and the other gets a delay, just like in real application, hence no effect.

- as a discrete event simulation, all the messages initial occurrence time is randomly generated for each simulation run. However, the following message arrivals are consistent with their periods. Two messages in the same node will not be allowed to be simultaneously generated. This is because in reality a node does not try to send two messages at the same time. It would organise all the ready to transmit messages in order in its buffer before sending them.
- CAN bus transmission speed is 1 Mbits/s which is the highest bit rate for automotive CAN use [25].

5.3.4 Simulation Models

The four architectures using CAN considered in this thesis were simulated. They are:

- Conventional centralised with limited CAN (*Architecture 2*)
- Total centralised (*Architecture 3*)
- Conventional centralised with functional integration (*Architecture 4*)
- Distributed wheel (*Architecture 5*) architectures

The signal Table B1 in Appendix B represents the simulated data for *Architecture 4* as an example. The departures from this example for each of the other architectures will now be examined.

5.3.4.1 Conventional Centralised with Limited CAN Interaction Architecture (*Architecture 2*)

As discussed earlier, *Architecture 2* is intended to represent the electronic architecture of current vehicles, which utilises CAN but to a limited extent. It is expected that CAN will be under more message load as more electronic systems and sensors/actuators are introduced or put on to the network. Since ECU connections to sensors and actuators of current vehicles are by hard wires, the sensor node and its related signals were taken off the vehicle signal list for *Architecture 2*. ABS, active suspension and 4WS control signals to their actuators were also taken off for the same reason.

The signals taken off from the list are number 2-11, 22-42, 57-64, 79-82.

5.3.4.2 Total Centralised Architecture (*Architecture 3*)

The architecture contains one centralised ECU in place of the ABS, active suspension and 4WS ECUs. The number of signals in the vehicle signal list is the same, but the three dynamic control ECU nodes are replaced with the centralised ECU node.

5.3.4.3 Conventional Centralised Architecture with Functional Integration (*Architecture 4*)

The simulation model of *Architecture 4* was constructed according to the vehicle signal list in Appendix B.

5.3.4.4 Distributed Wheel Controller Architecture (*Architecture 5*)

In *Architecture 5*, the ABS, active suspension and 4WS ECUs are replaced by four distributed wheel ECUs and the Central ECU. Sensor signals which are closely located to the distributed wheel ECUs were assigned to be transmitted by the ECUs. The distributed wheel controllers also took over the controls of ABS and suspension wheel actuators. The signal list modifications is shown in Table B2 in Appendix B.

5.3.4.5 Signal Bandwidth

The class C signals bandwidths and the overall bandwidths of all the architectures are given in Table 5.5. It can be seen that the *Architecture 2* has a lower designed bus bandwidth because of the use of additional hard-wired connections, as described above.

Architecture	Overall Bandwidth (kBits/s)	Class C Bandwidth (kBits/s)	Percentage of Class C Bandwidth
Architecture 2	186.8	153.6	82.2
Architecture 3	441.8	403.2	91.3
Architecture 4	441.8	403.2	91.3
Architecture 5	441.8	403.2	91.3

Table 5.5 Signal Bandwidth of all the Architectures

The detailed bandwidth distribution among the signals of *Architecture 3-4* is shown in Figure 5.25.

5.3.5 Simul8 Models

The *Simul8* model of *Architecture 4* is as shown in Figure 5.27, as an example.

Small scrolls on the left represent individual signals, that are from the list. The model is arranged such that the high priority signals are above the lower priority ones. 200 Hz signals have priority level 7, 100 Hz signals have priority level 6, 50 Hz signals have priority level 5, and so on. Note that for clarity, most of the 117 signals from the list are left out of the diagrams.

The scrolls and ECU images are for signal generation purpose in the simulation. The initial occurrence of each signal is firstly generated at a random start time by *Microsoft Excel.* Once the first signal is created, the subsequent signals of the same type follow according to their specified period. For example, as shown in Figure 5.28, the first vehicle speed signal is created at the random time of 2.52 ms from the start of simulation. Vehicle speed has a period of 10 ms, so the subsequent vehicle speed signals would arrive at 12.52, 22.52, 32.52 ms and so on from the starting time.

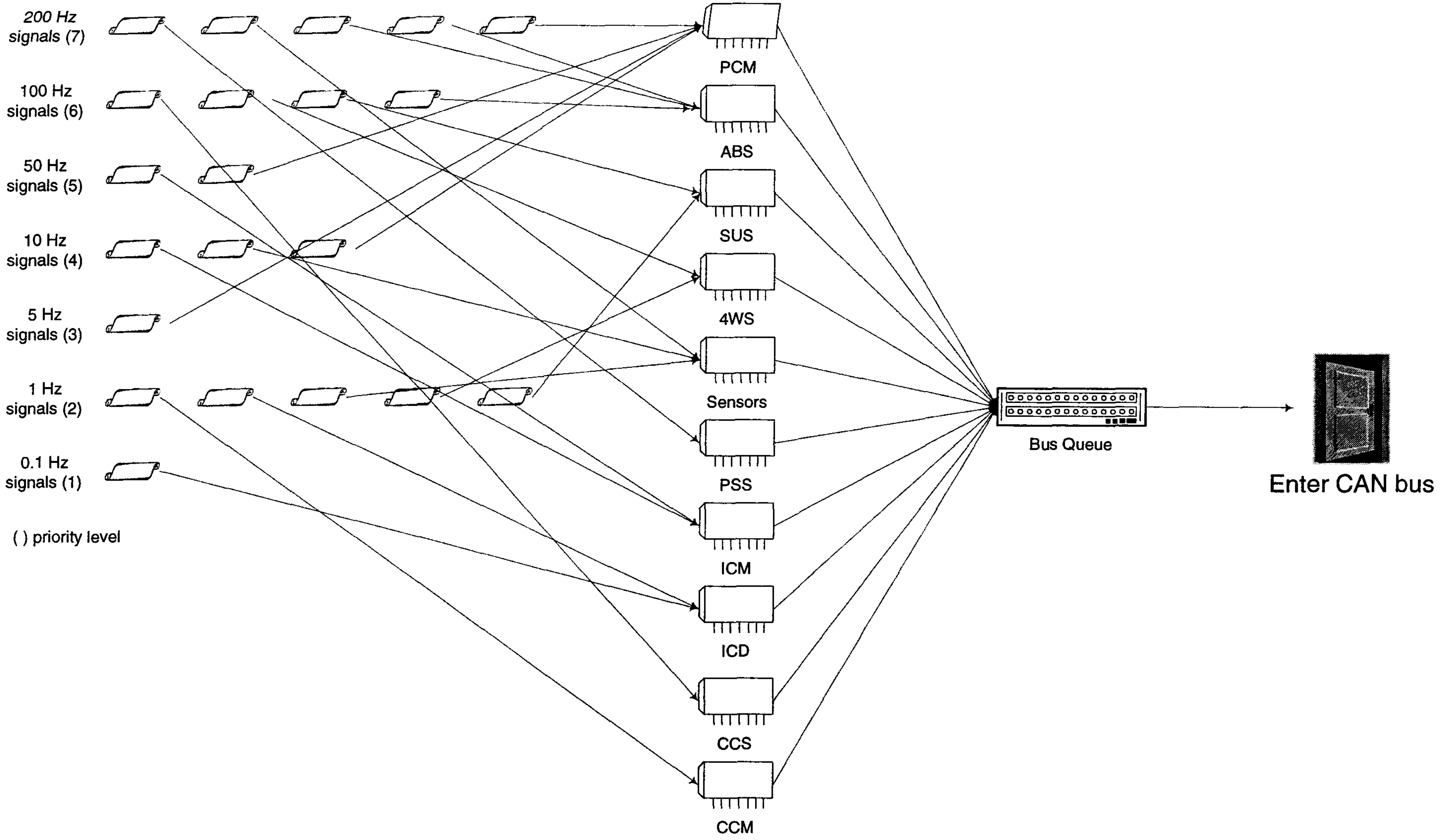


Figure 5.27 *Simul8* model of *Architecture 4*

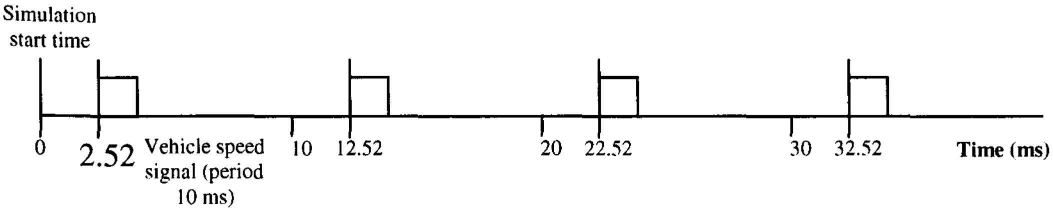


Figure 5.28 Timing diagram of a 10 ms period signal generation

As indicated by arrows, these signals go to the originating ECU to which they belong, to be transmitted onto the CAN bus. The whole process described above is equivalent to ECUs creating signals to be sent onto the CAN bus, in real vehicle applications.

From ECUs linked by arrows to Bus Queue represents data queuing to be transmitted from ECUs onto the CAN bus. An ECU will let its signals join a queue one by one, equivalent to an ECU attempting to transmit one signal at a time. More than one signals in the Bus Queue at a time symbolises message collision. The Bus Queue arranges incoming signals in order according to their priority, equivalent to CAN message contention. The highest priority signal is put in front of the queue. The Bus Queue then let the highest priority signal onto the CAN bus (displayed as a door image) once the bus is free. Each signal occupies the CAN bus for 64 ms, equal to the time taken to transmit the 8 byte message.

The simulation models representing the 4 different electronic architectures are essentially of the same format, except for the difference in the number of ECUs and signals, and their signal routing.

5.3.6 Simulation Run

A simulation was run for an equivalent of 1 real-time second at time. One second covers the periods of all the signals except for signal no. 117, whose period is 10s and hence of little significant to the CAN bus load. Since all the signals are assumed periodic, any longer simulation run would give a repetitive result to the 1 second run.

For each architecture, the simulation was run 100 times with different sets of random numbers. Each simulation was run for 1 real-time second. A number of simulations were run in order to simulate different possibility of messages arriving on the CAN bus at different times. Each simulation involves 4800-6800 messages getting access to the CAN bus. The time which the two groups of class C signals (of period 5 and 10ms), which are for real-time control, wait in the CAN Bus Queue plus the transmission time was collected. This is equivalent to the signal time delay associated with CAN in real applications.

5.3.7 Results and Analysis

The frequency distribution of the CAN delay of the two groups of class C signals are shown in the graphs below.

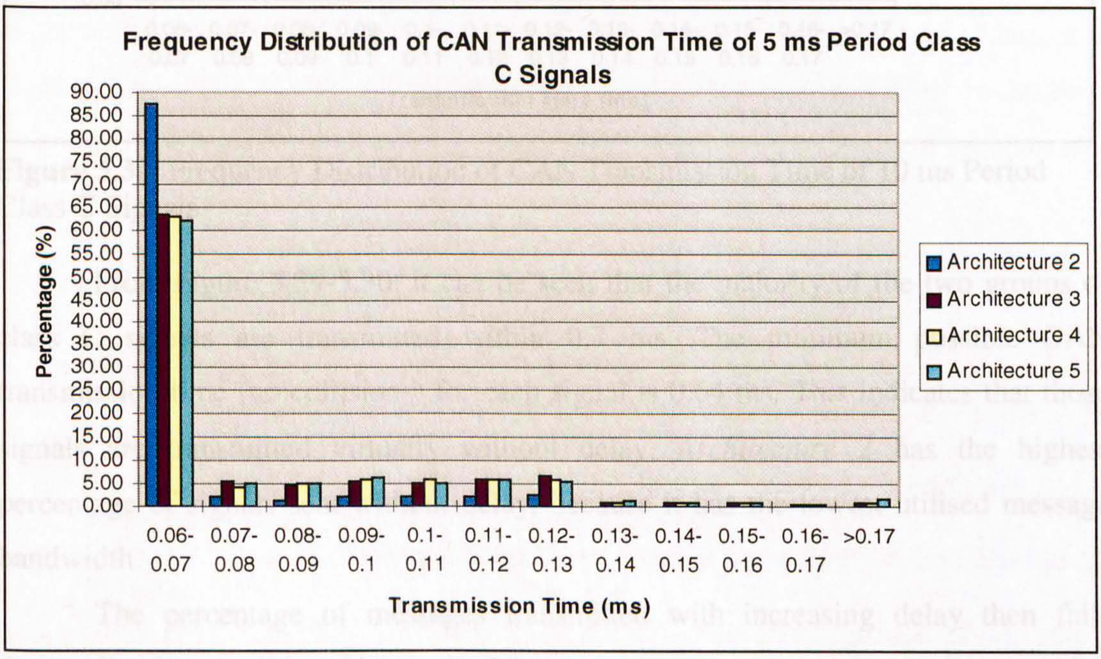


Figure 5.29 Frequency Distribution of CAN Transmission Time of 5 ms Period Class C Signals

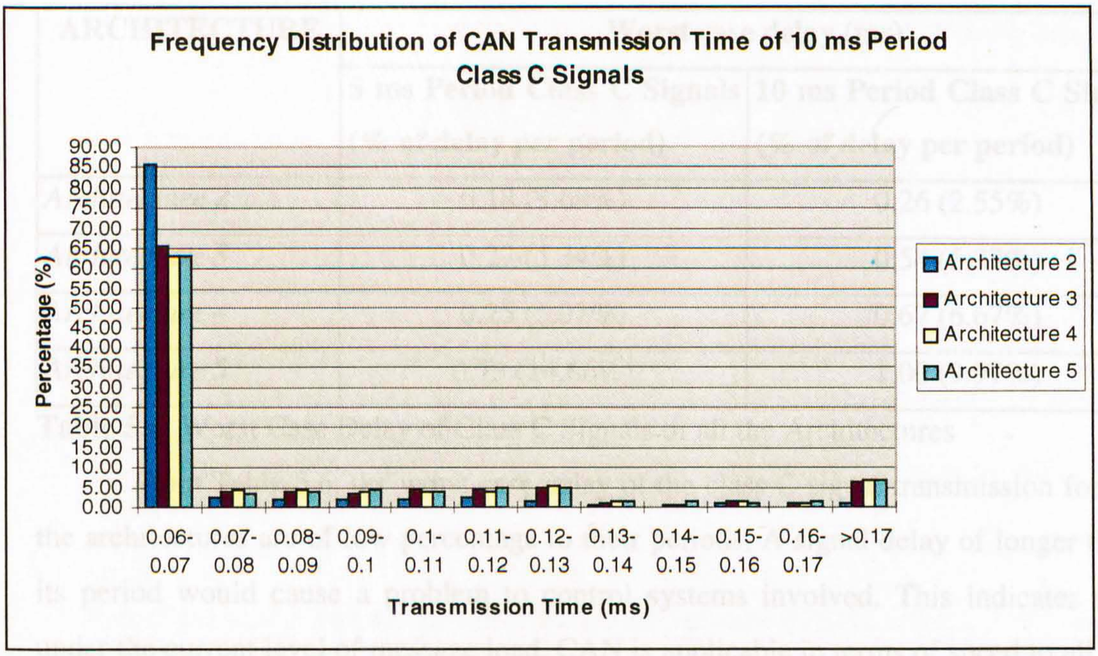


Figure 5.30 Frequency Distribution of CAN Transmission Time of 10 ms Period Class C Signals

From Figure 5.29-5.30, it can be seen that the majority of the two groups of class C signals are transmitted within 0.7 ms. The minimum possible CAN transmission time (no collisions) for each signal is 0.64 ms. This indicates that those signals are transmitted virtually without delay. *Architecture 2* has the highest percentage of signals sent without delay, because it has the lowest utilised message bandwidth.

The percentage of messages transmitted with increasing delay then falls drastically for all the architectures. However, the larger number of 10 ms period signals have experienced long delays than the 5 ms periods signals. This can be seen from the higher percentage of signals with transmission time longer than 17 ms (the last bars on the chart) in Figure 5.30 than those in Figure 5.29. Also the 10 ms period signals have experienced longer worst case delay than the 5 ms signals. This could be expected, since the 10 ms period signals have lower priority than the 5 ms signals. The worst case delays of all the architectures are shown in Table 5.6.

ARCHITECTURE	**Worst case delay (ms)**	
	5 ms Period Class C Signals (% of delay per period)	**10 ms Period Class C Signals (% of delay per period)**
Architecture 2	0.18 (3.69%)	0.26 (2.55%)
Architecture 3	0.27 (5.34%)	0.54 (5.37%)
Architecture 4	0.25 (5.07%)	0.67 (6.67%)
Architecture 5	0.73 (14.66%)	1.00 (9.97%)

Table 5.6 Worst Case Delay of Class C Signals of all the Architectures

From Table 5.6, the worst case delay of the class C signal transmission for all the architectures are of low percentage to their periods. A signal delay of longer than its period would cause a problem to control systems involved. This indicates that under the current level of message load, CAN is applicable in terms of speed to all the electronic architectures.

5.4 References

1. *Rover* **Range Rover Electrical Manual** Rover Group 1995
2. *Ribbens W* **Understanding Automotive Electronics 4th Edition** McGraw Hill, 1995
3. *Philips Components* **Integrated Circuits Data Handbook IC14 Microcontrollers NMOS, CMOS,** Philips 1989
4. *Schmerler S, et al.* **Towards Real-Time System Specification and Design** SAE 961631
5. *Dieterich K* **Methods and Tools for the Efficient Development of Automotive Electronics** SAE 950571
6. *Maclay D* **Simulation gets into the Loop** IEE Review, May 1997
7. *Glibbery R* **Suspension Ride Height Controller** Lucas Automotive Ltd., 1990
8. *Motorola* **Selecting the Right Microcontroller Unit** Motorola, 2000
9. *Lipovski G* **Introduction to Microcontrollers Architecture, Programming, and Interface for the Motorola 68HC12** Academic Press, 1999
10. *Bannatyne R* **Selecting a Microcontroller** Embedded Systems Programming Vol.11, Issue 4, 1998
11. *Comer D* **Microprocessor-Based System Design** CBS College Publishing, 1986
12. *Spasov P* **Microcontroller Technology The 68HC11 3rd Edition** Prentice Hall, 1999
13. *Aoyama Y, et al.* **Development of the Full Active Suspension by Nissan** SAE No.901747
14. *Masutomi S, et al.* **Development of ABS and Traction Control Computer** SAE No.901707
15. *Upender B* **Analysing the Real-Time Characteristics of Class C Communications in CAN Through Discrete Event Simulations** SAE No.940133

16. *Eguchi T, et al.* **Development of "Super Hicas", a New Rear Wheel Steering System with Phasereversal Control** SAE No.891978
17. *Yokoya Y, et al.* **Integrated Control System Between Active Control Suspension and Four Wheel Steering for the 1989 CELICA** SAE No.901748
18. *Hawley G* **Selecting a Real-Time Operating System** Embedded Systems Conference Papers, Miller Freeman, 1999
19. *Cahners Business Information* **Table of Commercial Embedded RTOSes** EDN Magazine website http://www.ednmag.com/ednmag/extras/embeddedtools/rtosdisplay.asp 2001
20. *Jurgen R* **Automotive Electronics Handbook 2nd edition** McGraw-Hill, 1999
21. *Lemieux J* **Moving Efficiently from Assembly Language to C** Embedded Systems Conference, 1999
22. *Lawrence P, et al.* **Real-Time Microcomputer System Design: An Introduction** McGraw-Hill, 1987
23. *Paccard E* **Technology for a New Automotive Era** Real-Time Magazine Issue 3, 1999
24. **Electrical and Vacuum Trouble Shooting Manual FPS-12119-93: 1993 Town Car** Ford Motor Company, 1993
25. *Bosch* **CAN Specification Version 2.0** Robert Bosch GmbH, 1991
26. *Toeppe S, et al.* **Commercial RTOSes for Automotive Applications** Embedded Systems Programming, July 2000
27. *Stepner D, et al.* **Embedded Application Design Using a Real-Time OS** Proceedings of 1999 Design Automation Conference, 1999
28. *O'Dowd D* **Real-Time Operating Systems Traget Missions-Critical Embedded Systems** Real-Time Magazine, July-September 1997
29. *Keate L* **A Real-World Approach to Benchmarking DSP Real-Time Operating Systems** Proceedings to WESCON, 1997
30. *Tribolet C, et al.* **Embedded C and C++ Compiler Evaluation Methodology** Embedded Systems Conference, 1999
31. *Shooman M* **Software Engineering: Design, Reliability, and Management** McGraw-Hill, 1983
32. *Kernighan B, et al.* **The C Programming Language 2nd Edition** Prentice-Hall, 1988
33. *Tindell K* **Embedded Systems in the Automotive Industry** Embedded Systems Conference, 1999
34. *Motorola* **Master Selection Guide** Motorola, 1999
35. *Trialog* **OSEKtr 2.0 Managing the Growth of Automotive Electronics** Trialog, 2000
36. *Wind River* **Tornado for OSEKWorks 2.0** Wind River, 2000

CHAPTER 6

ELECTRONIC ARCHITECTURE EFFECTS ON COMPLEXITY AND COSTS

An electronic control system consists of four parts, sensors, actuators, an ECU, and wiring harness. This chapter compares the five architectures' effects on system component complexity and costs. The focus will be upon wiring harness and ECU, since it is these that vary between architectures. The number of wires in each architecture will first be estimated, together with their weight. The microcontroller requirements for each ECU are estimated in the previous chapter. The number of other ECU components will be approximated, thus combined with the microcontroller specifications to form a measure of ECU complexity. The cost of each system part is researched and the overall system component cost of each architecture is calculated. The comparison between the cost of each architecture is made, and the effects of the future trends and other factors on costs are discussed.

6.1 Effect of the Vehicle Wiring Harness on Different Architectures

The complexity of wiring harness is an important consideration in the design and assembly of any electronic control system, for its cost, weight and ease of vehicle assembly. Tighter safety regulations and higher performance demands from the market result in the increased electronic control system complexity. As the electronic systems grow more complex, the number of ECU functions, and inputs and outputs invariably increases. This is confirmed in the case of engine management and powertrain ECUs, as shown in Figure 6.1 [1]. The increasing number of input and output signals, needed in automotive electronic control systems, underlines the growing significant of the vehicle wiring harness.

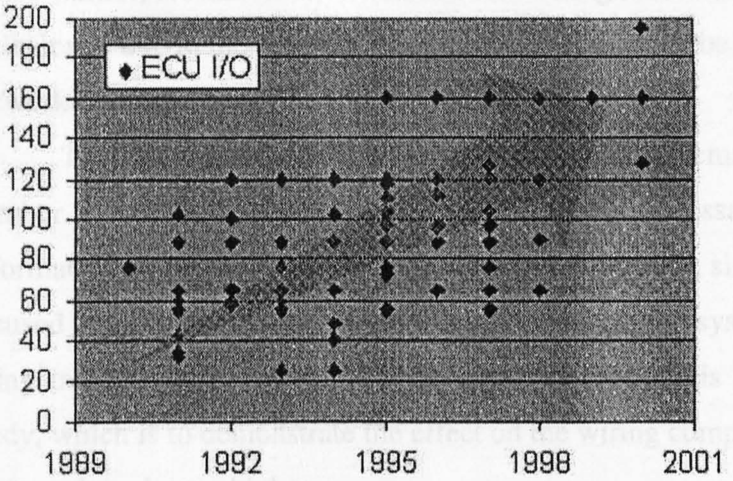


Figure 6.1 Growth in the number of powertrain ECU I/Os against calendar year

The increase in wiring harness complexity also results in the increase in the number of connectors. Together, these two are the major source of failures in vehicle electronics [2]. Furthermore, the greater the amount of wiring harness carried, the more vehicle weight problems, Electromagnetic Interference (EMI) related, and physical assembly problems when fitted to the vehicle. A study has also shown that wiring is the second most expensive item in modern vehicles [3].

It was in order to overcome the above wiring related problems, that multiplexing technology was originally introduced. A large number of networking systems have since been developed by car manufacturers and suppliers to tackle these wiring problems.

Having emphasised the impact of wiring harness on the design of electronic control systems, this chapter is dedicated to analysing each electronic architecture's wiring characteristics in terms of wire count and length.

6.1.1 Wiring Models of Dynamic and Powertrain Control Systems

Dynamics and powertrain control systems form the subject of this thesis. To form a wiring model of each electronic architecture, exemplary wiring of each control system has to be selected. The wiring models of these systems will be used as bases for all the electronic architectures. Power distribution wiring will not be included, being dissimilar to signal wiring and independent of networking protocols.

The three chosen dynamic control systems, namely ABS/ASR, active suspension and four wheel steering, are included in the wiring model. Powertrain control systems, which comprise Engine Management System (EMS), transmission, cruise control, are also used to construct the wiring models, as is power steering. Their inclusion is due to the fact that these systems would also be likely to use a high speed networking bus.

The functionality of the exemplary control systems are taken from a small number of published papers, which provide either necessary wiring or signal flow information [4-14]. In case of powertrain control systems, since their functions are not focused upon in this thesis, the most complicated current systems found are chosen, as being indicative of mainstream near-future systems. This is due to the aim of this study, which is to demonstrate the effect on the wiring complexity of an electronically advanced modern vehicle.

The exemplary active suspension and four wheel steering systems, used in Chapter 3 on vehicle electronic simulation, are also employed in the wiring model. The exemplary 4WS was designed by the manufacturer to have an integrated control with an ABS when wheel slip occurs. However, these two systems are treated individually here, since it is assumed that each system would be provided separately by different suppliers.

The ABS control model used for simulation in Chapter 3 is derived from a Bosch ABS description [4]. Its signal flow or wiring information is not available. Therefore, a different ABS, which has its signal flow data elaborately published, is used in the vehicle wiring model instead [5]. This system is a combined ABS and Traction Control (ASR), which is advanced and gaining popularity among top end vehicles, so the ASR is included in the model.

Using all the information on signals and wiring of all the above control systems, a vehicle signal table is derived as shown in Table C1 in Appendix C [5-12]. The table is sorted into individual system inputs and outputs. The sources and destinations of the input and output signals, respectively, are also given in the table.

The numbers associated with the sources or destinations are used to identify sensors or actuators in the wiring models in Figures 6.2 to 6.5. It should be noted that the engine management system is very complex and includes a large number of engine sensors and actuators, such as knock and oxygen sensors, fuel injectors, etc. However, these devices are within the engine proximity or actually situated inside the engine, the signal flow between the engine sensors and actuators, and the EMS ECU is treated as internal to the engine management system. Hence it is omitted from the model.

In Figure 6.2-6.5, system components which involve a lot of signal types have their names written. Sensors or actuators which have fewer wires connected to are given identifying numbers, in order to save diagram space. The list of the components and their associated numbers is displayed in Table 6.1.

	Vehicle Diagram Component Number
	Sensors and Switches □
1	throttle position sensor
2	ignition key
3	vertical G sensors x3
4	lateral G sensors x2
5	height sensors x4
6	longitudinal G sensor
7	speed sensor
8	door switch
9	wheel speed sensors x4
10	kick down switch
11	gear lever position
12	brake fluid level switch
13	hand brake switch
14	brake switch
15	steering wheel angular velocity sensor
16	steering wheel torque sensor
17	steering wheel angle sensor
18	yaw rate sensor
19	4WS oil level sensor
	Actuators ○
1	suspension wheel pressure control valves x4
2	throttle actuator
3	sub-throttle actuator
4	4WS motor
5	power steering unit
6	ABS wheel actuators x4

Table 6.1 List of sensors and actuators in the models

The following section describes the wiring models of each vehicle electronic architecture.

In the wiring models, ECU and component locations are drawn according to the information available, so that the length of system wiring is visually representative. However, due to limited diagram space, lack of information on a small number of components, and component simplifications, some components are located accurately only up to the appropriate zone in a vehicle. For example, the suspension hydraulic system components, such as pumps and valves which spread throughout the underbonnet zone. For simplification, all components are treated as a single block situated in that zone . This is not entirely accurate, but any errors introduced would tend to cancel each other out (i.e. some wires would be longer than the approximation, some shorter), so its connection to the active suspension ECU still provides a good appreciation of the wire length required. The ABS exemplary system does not provide information on the ABS ECU location. The ABS ECUs can be placed in the engine or passenger compartments [7]. In order to be consistent with the active suspension and 4WS ECUs, the ABS ECU is assumed to be passenger compartment (zone) mounted.

Smart sensors and actuators, which employ intelligent sensor and actuator nodes in an area such as the underbonnet zone to handle their information transfer, are not shown in the modelling. This is because the modelling process is intended to compare the amount of wiring among studied architectures, within the same criteria. Since the conventional centralised architecture is incapable of such technology, it is therefore omitted. However, smart sensors and actuators are also believed to yield benefits in terms of wiring reduction to all the architectures with networking capability.

6.1.1.1 Conventional Centralised Architecture (*Architecture 1*)

All the vehicle electronic architectures are described in chapter 2. The Conventional Centralised architecture has each electronic system as a standalone. ECU interaction is very limited, except for some sensor signals such as vehicle speed, which may be shared among a few control systems. No form of networking exists. All the connections within or between electronic control systems, are by point to point wiring.

The wiring model of the Conventional Centralised architecture is shown in Figure 6.2.

6.1.1.2 Conventional Centralised with Limited CAN Architecture (*Architecture 2*)

This architecture has similar structure to the previous *Architecture 1*, except for the introduction of CAN in the vehicle. The CAN network provides the means of communication between ECUs. All the information transferred between ECUs, previously done via individual wires, is now carried out through a two wire CAN network. Figure 6.3 displays the Conventional Centralised with Limited CAN architecture.

6.1.1.3 Total Centralised Architecture (*Architecture 3*)

The Total Centralised architecture has a single centralised dynamic control ECU, which performs all the control tasks of ABS/ASR, active suspension, power steering and 4WS. A single EMS ECU is also used to control all the engine and powertrain applications, in effect combining engine management, transmission and cruise control ECUs into one. The CAN bus is applied as a means of data transfer between the centralised dynamic control, EMS and other ECUs. The Total Centralised architecture model is shown in Figure 6.4.

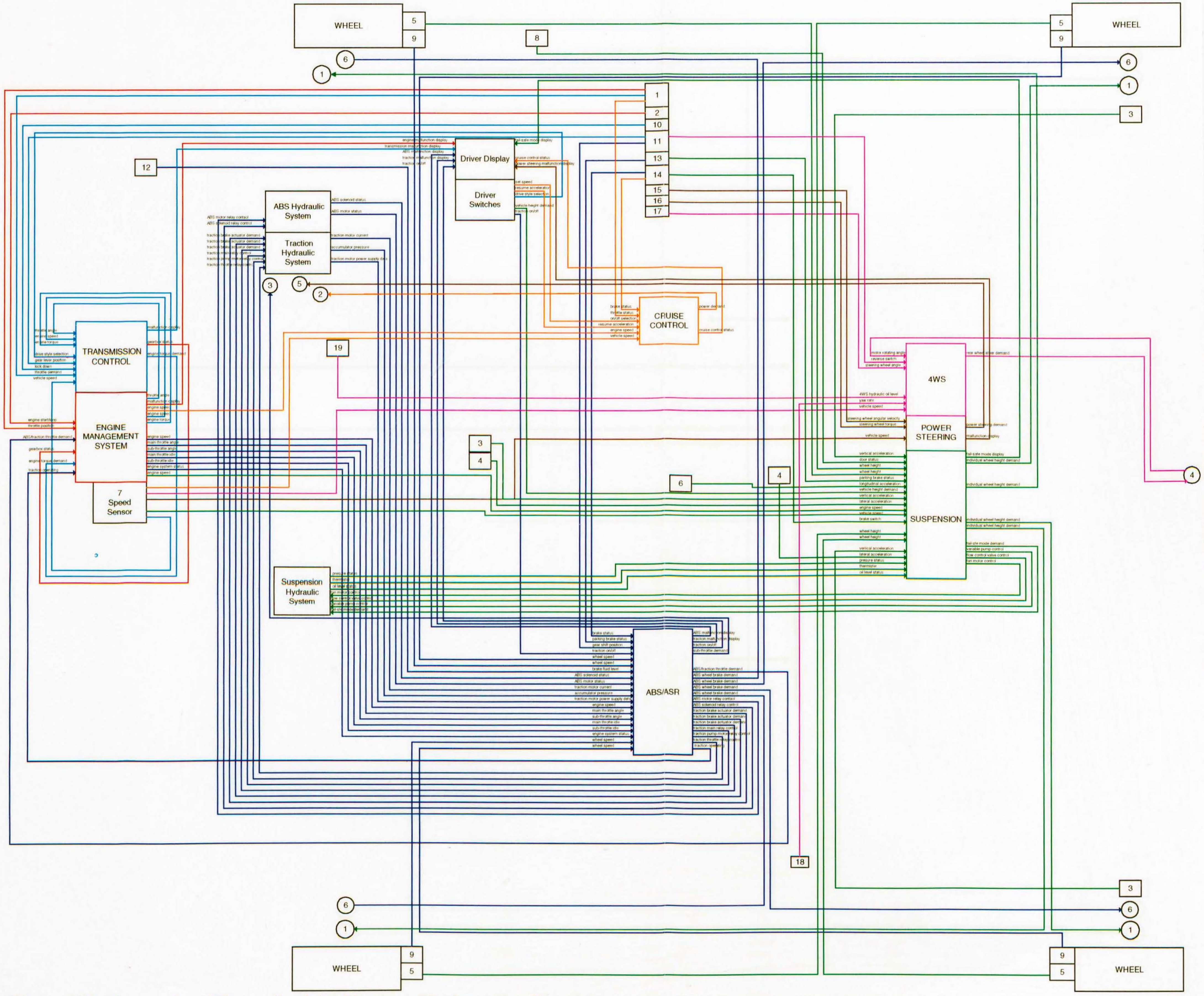
6.1.1.4 Conventional Centralised with Functional Integration Architecture (*Architecture 4*)

Without the applications of smart sensor and actuator nodes, the architecture has the same wiring arrangement as *Architecture* 2,. Hence Figure 6.3 also represents the wiring of this architecture.

6.1.1.5 Distributed Wheel Controller Architecture (*Architecture 5*)

In this Distributed Wheel architecture, four individual wheel control ECUs are located near all four wheels. Each is responsible for controlling the ABS and suspension of its corresponding wheel. A central controller in the vehicle acquires and distributes sensor data which needs to be shared among individual wheel ECUs. The sensor and actuator signals for each wheel, exclusive to each individual distributed wheel ECU such as wheel height, are transferred to the ECU via individual wire links. The integrated engine and powertrain ECU remains the same as in *Architecture 3*, while 4WS and power steering ECUs are integrated with the Central ECU. The CAN network is used for data sharing among all the ECUs.

The wiring model of the Distributed Wheel architecture is as shown in Figure 6.5.



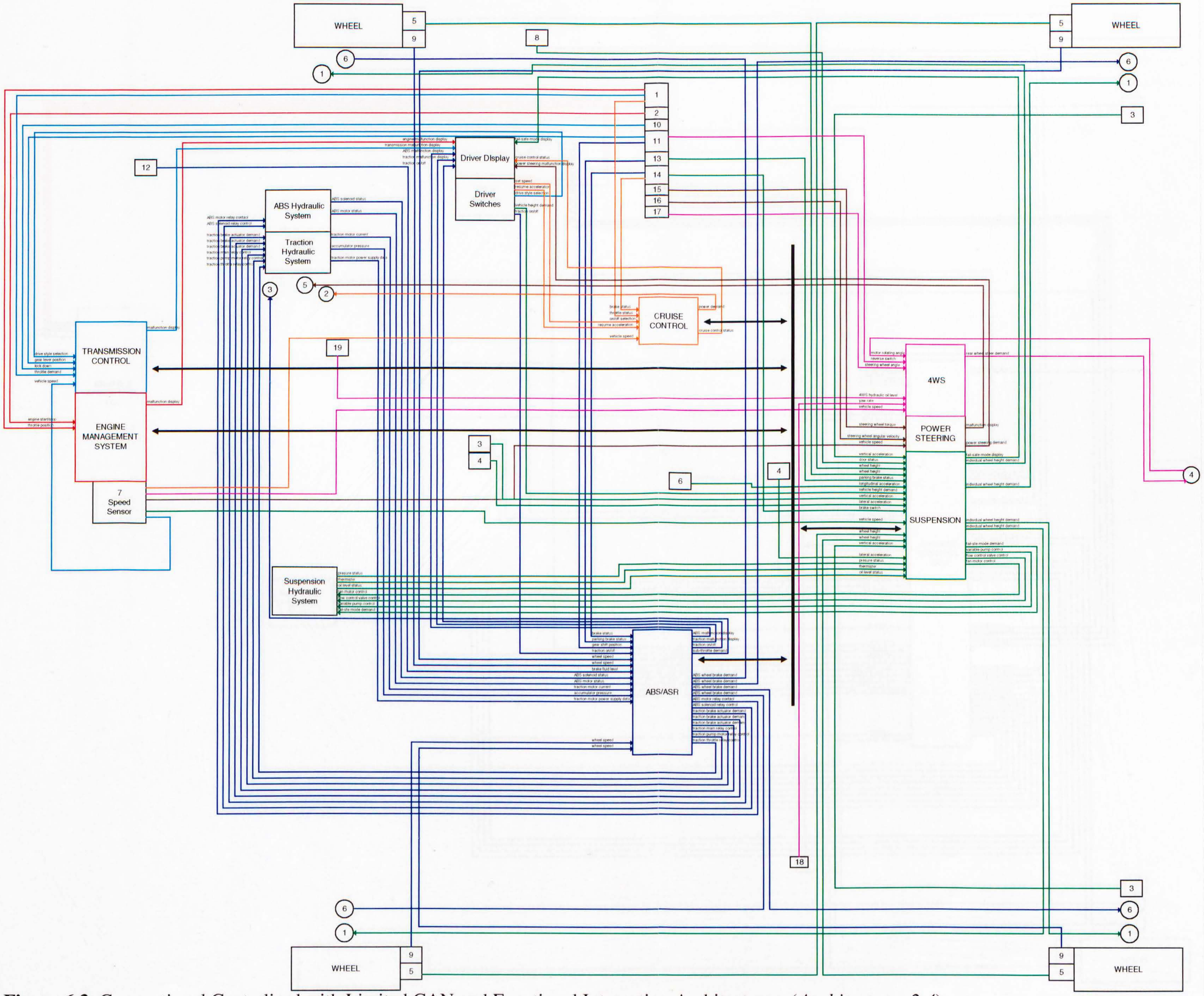
Sensors and Switches (squares)

1	throttle position sensor
2	ignition key
3	vertical G sensors x3
4	lateral G sensors x2
5	height sensors x4
6	longitudinal G sensor
7	speed sensor
8	door sensors x4
9	wheel speed sensors x4
10	kick down switch
11	gear lever position
12	brake fluid level switch
13	hand brake switch
14	brake switch
15	steering wheel angular velocity sensor
16	steering wheel torque sensor
17	steering wheel angle sensor
18	yaw rate sensor
19	4WS oil level sensor

Actuators ○

1	suspension wheel pressure control valves x4
2	throttle actuator
3	sub-throttle actuator
4	4WS motor
5	power steering unit
6	ABS wheel actuators x4

Figure 6.2 Conventional Centralised Architecture (*Architecture 1*)



Sensors and Switches (squares)

1	throttle position sensor
2	ignition key
3	vertical G sensors x3
4	lateral G sensors x2
5	height sensors x4
6	longitudinal G sensor
7	speed sensor
8	door sensors x4
9	wheel speed sensors x4
10	kick down switch
11	gear lever position
12	brake fluid level switch
13	hand brake switch
14	brake switch
15	steering wheel angular velocity sensor
16	steering wheel torque sensor
17	steering wheel angle sensor
18	yaw rate sensor
19	4WS oil level sensor

Actuators ○

1	suspension wheel pressure control valves x4
2	throttle actuator
3	sub-throttle actuator
4	4WS motor
5	power steering unit
6	ABS wheel actuators x4

Figure 6.3 Conventional Centralised with Limited CAN and Functional Integration Architectures (*Architectures 2,4*)

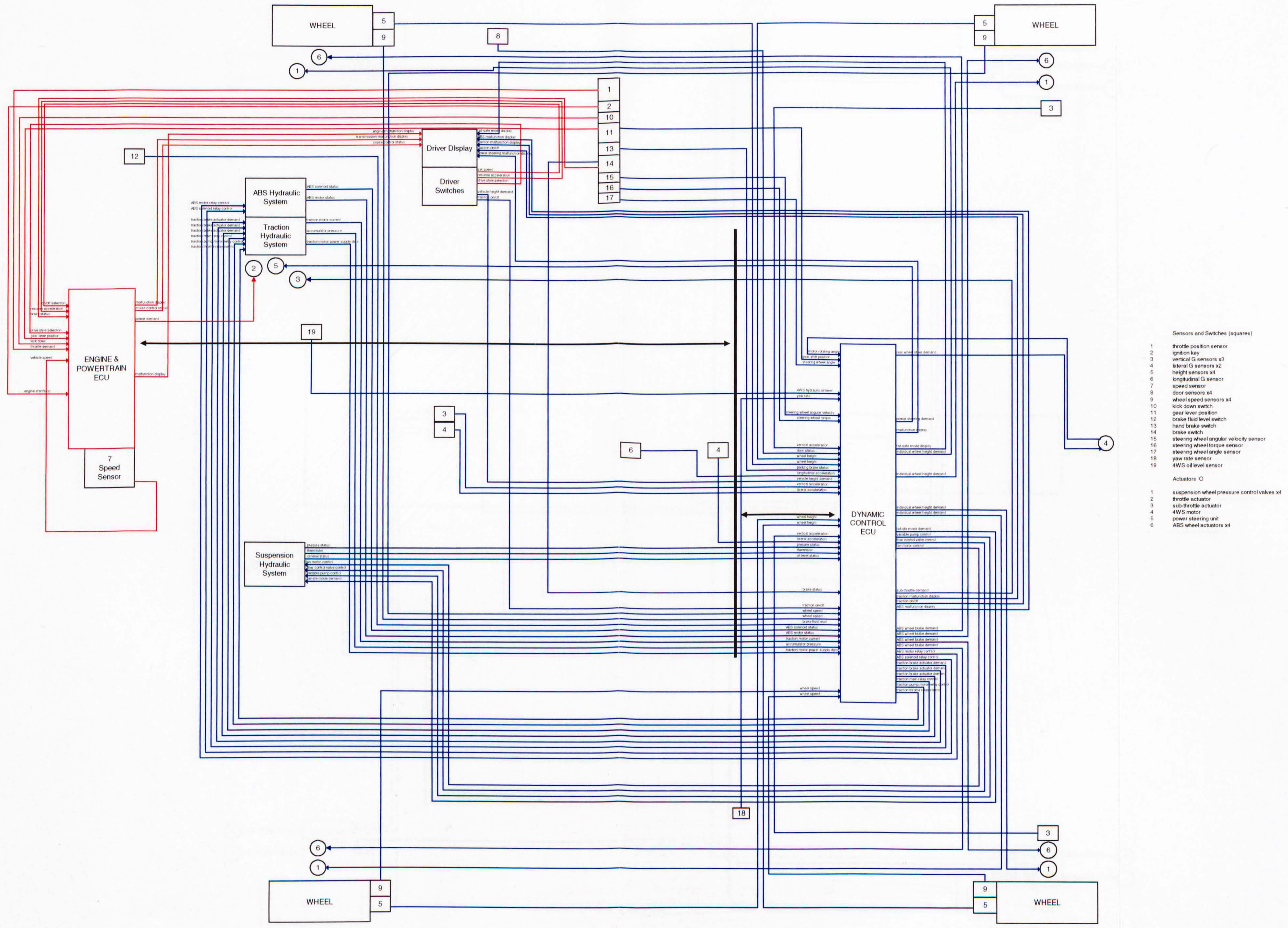


Figure 6.4 Total Centralised Architecture (*Architecture 3*)

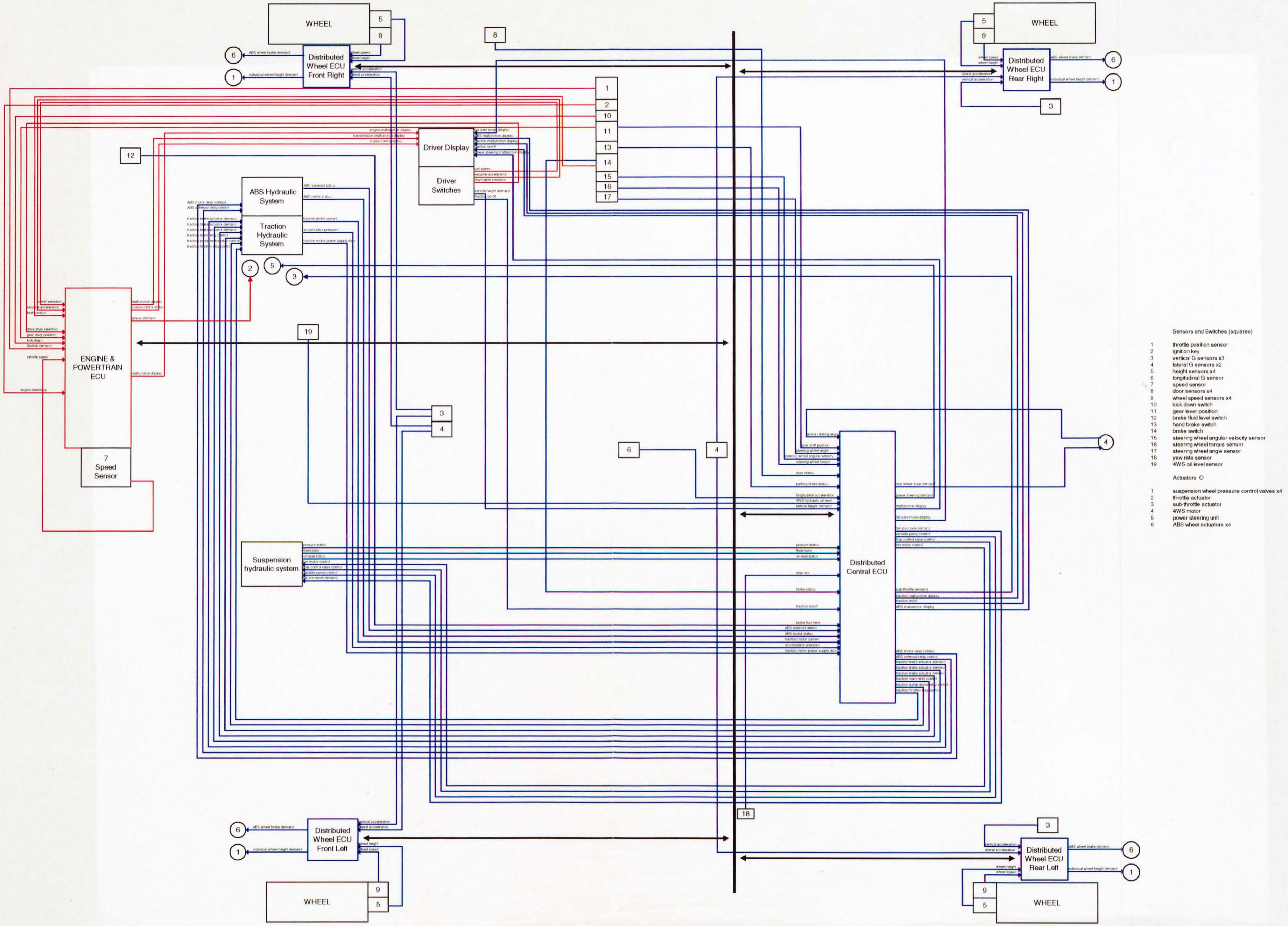


Figure 6.5 Distributed Wheel Controller Architecture (*Architecture 5*)

6.1.2 The Number of Wires and Their Estimated Total Length and Weight of Each Architecture

The number of wires, their estimated total lengths and weights will be used for comparison and discussion of the different vehicle electronic architectures.

6.1.2.1 The number of Wires in Different Electronic Architectures

The total number of wires, in each of the 4 vehicle electronic architecture models, are simply counted and shown in Table 6.2.

Vehicle Electronic Architectures	Total Number of Wires
Conventional centralised architecture *(Architecture 1)*	100
Conventional centralised with limited CAN and with functional integration architectures *(Architectures 2 and 4)*	85
Total centralised architecture *(Architecture 3)*	76
Distributed wheel architecture *(Architecture 5)*	79

Table 6.2 Total number of wires in different vehicle electronic architectures

6.1.2.2 The Estimated Total Lengths of Wires in Different Electronic Architectures

The total wire length of each architecture is estimated by first splitting the wires into groups according to their lengths. This is done in relation to an estimated distance between the components at the two ends of a wire. The length of each wire group is estimated, as will be described later. The total wire length for each architecture can then be added up from all the wires.

As stated at the beginning of this chapter, that in the models, the EMS and transmission control ECU are located in the underbonnet zone, together with the vehicle speed sensor, ABS and traction actuators, and the suspension hydraulic system. The other five ECUs are situated under the passenger seats, classified as in the centre zone. All the driver control switches and display are obviously around dashboard zone. The ABS and suspension wheel actuators are in proximity to the wheels, and classified as in wheelarch zones. Finally, the 4WS actuator is located near the rear axle and considered as in boot zone. The wiring models are drawn according to these classifications.

Figure 6.6 shows the zonal classification of a vehicle.

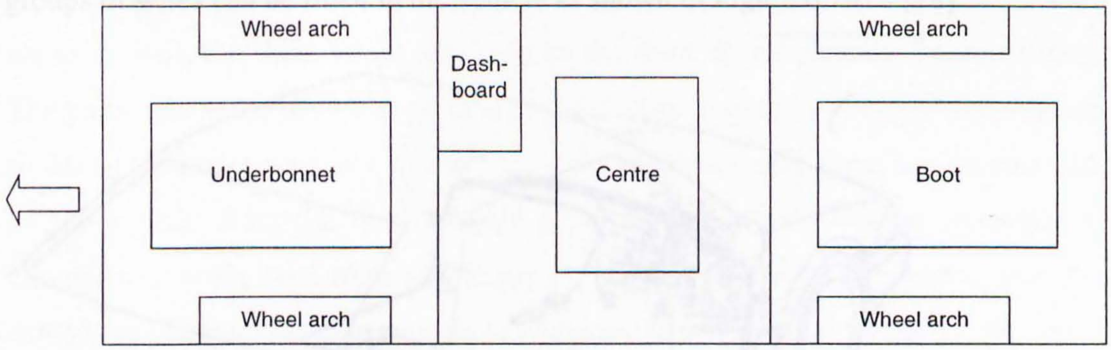


Figure 6.6 Vehicle zonal diagram

Based on the component locations and their relative distance apart specified above, looking at the set of wires in the four models, all the wires can be categorised into 4 length groups, according to the zones of the two components that they link as follows:

- centre-underbonnet/boot zones, for example, the wire that links the engine speed signal between EMS and the cruise control ECU, or the rear wheel steer signal wire between the 4WS ECU and the 4WS actuator.
- dashboard-underbonnet/centre zones, for example, a driver display signals from EMS or the active suspension ECU to the driver.
- centre-wheelarch zones, for example, a wheel height signal between the active suspension ECU to an active suspension actuator
- same zones, such as one that carries engine speed signal between EMS and the transmission control ECU.

From the above wire grouping definition, all the wires in each wiring architecture can be classified in to groups as shown in Table 6.3.

Vehicle Electronic Architectures	**Number of Wires**				
	Centre-underbonnet/ boot	**Dashboard-underbonnet/ centre**	**Centre-wheelarch**	**same zones**	**Total**
Architecture 1	41	35	18	6	100
Architectures 2,4	31	35	18	1	85
Architecture 3	26	30	18	2	76
Architecture 5	26	27	6	20	79

Table 6.3 Wiring length classification of different vehicle electronic architectures

From a wiring inspection of a typical medium/large saloon car [15], the four groups of wires can be fitted in the vehicle as shown in Figure 6.7-6.8 [18].

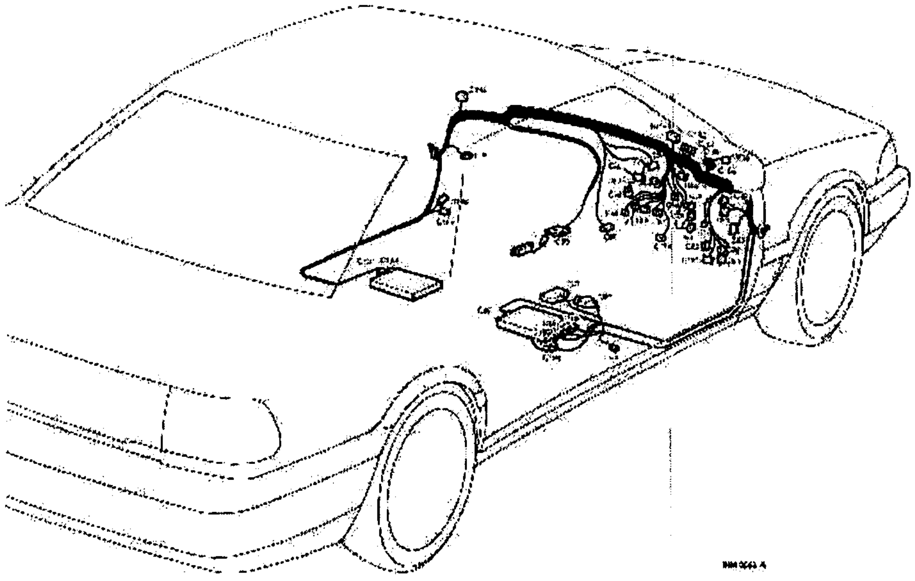


Figure 6.7 ECU wiring from passenger compartment to door panels

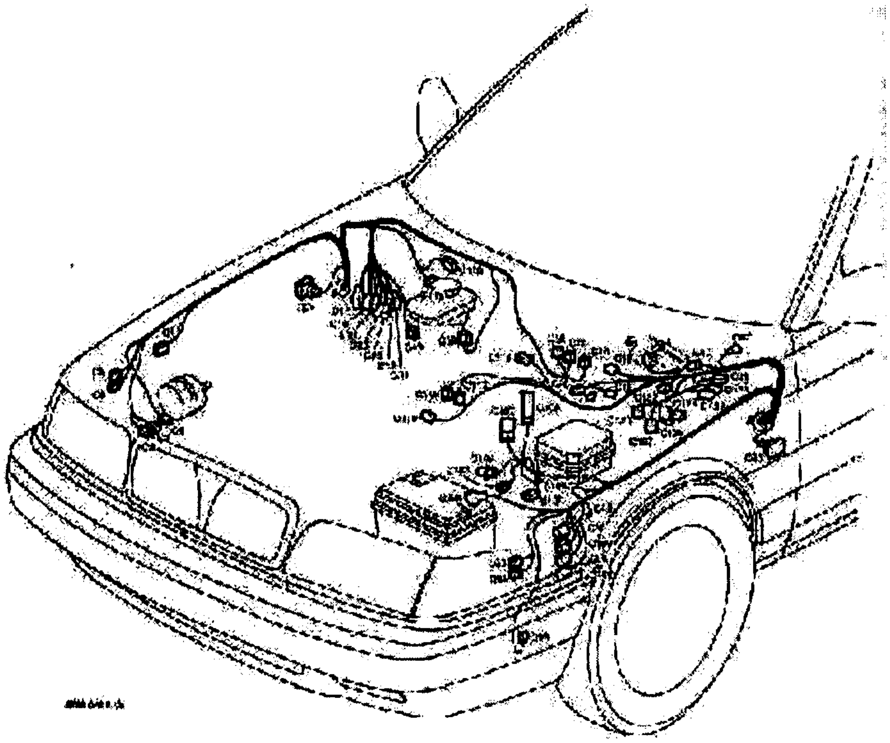


Figure 6.8 ECU wiring from passenger compartment to engine bay area

Figure 6.7 demonstrates how the ECUs in the centre zone are wired. All the wires from these ECUs are first led in bundles, to the sides of the vehicle along the floor. From there the wires are laid towards the front or rear of the vehicle, depending on the locations of end components, along the sides of the vehicle at floor level below the doors.

Figure 6.8 shows how these wires are linked to the underbonnet zone. They are laid along the floor below the doors to the front of the passenger compartment. The parts, where the wires can go though the chassis from the passenger compartment to the underbonnet zone, are located below the dashboard and glove box on each side of the vehicle. Through these chassis gaps, the wires can then be connected to components in the front wheelarch zones or led along the front wing panels and then across the underbonnet zone to link to components in that area. Similarly, the wires, linked to the rear wheelarch or boot zones, are laid in this manner.

From the above wiring information, the lengths of wires belonging to each of the four groups can be estimated as shown below.

The estimated length of a wire that link two components between:

Centre - underbonnet/boot zones – total length is the sum of the lengths of the following sections of wiring between:

component in centre zone – vehicle side ≅ 0.5 vehicle width (W)

vehicle side – front/rear end of passenger compartment (along door panel) ≅ 0.5 wheelbase (WB)

front/rear of passenger compartment (floor level) – chassis gaps linked to engine bay/boot zone (below dashboard level) ≅ 500 mm

chassis gaps - centre of underbonnet/boot zone ≅ 0.5W

Total length ≅ 0.5W + 0.5WB + 500 + 0.5W = 1W + 0.5 WB + 500 mm

Dashboard - centre/underbonnet zones - total length is the sum of the lengths of the following sections of wiring between:

component in centre zone – vehicle side ≅ 0.5W

vehicle side – driver's position (along door panel) ≅ 0.25WB

driver's position – dashboard (vertical wiring) ≅ 500 mm

Total length ≅ 0.5W + 0.25WB + 500 mm

Centre - wheelarch zones - total length is the sum of the lengths of the following sections of wiring between:

component in centre zone – vehicle side ≅ 0.5W

vehicle side – front/rear end of passenger compartment (along door panel) ≅ 0.5WB

front/rear of passenger compartment (floor level) – chassis gaps linked to underbonnet/boot zones (below dashboard level) ≅ 500 mm

Total length ≅ 0.5W + 0.5WB + 500 mm

Short distance – wires that link two components in the same zones are given an approximate length of 500 mm each.

In order to quantify the above estimated wire lengths, wheelbase and width of a vehicle are needed. These dimensions are taken from two types of cars: a large, luxury saloon and a compact car. This allows the wire lengths of different architectures to be compared, when applied to two passengers on different market ends. The comparison could indicate which architectures are best suited to which type of vehicles. The required dimensions are taken from those of the Mercedes-Benz S-class and the Vauxhall Corsa, both of which are well known among their classes. Their dimensions are [16,32]:

Mercedes-Benz S-class:	Width = 1.855 m	Wheelbase = 3.085 m
Vauxhall Corsa:	Width = 1.610 m	Wheelbase = 2.440 m

The lengths of the four types of wires can now be estimated according to the formula derived earlier. Each type of wire length, together with the total wire length of each electronic architecture, when installed in each vehicle are shown in Table 6.4.

Vehicle Electronic Architectures	**Length of Wires (m) (Mercedes-Benz S-class / Vauxhall Corsa)**								
	centre-underbonnet /boot		**dashboard-underbonnet /centre**		**centre-wheelarch**		**same zones**		**Total Length**
	No.	**length**	**No.**	**length**	**No.**	**length**	**No.**	**length**	
Architecture 1	41	159.8/ 136.5	35	77.0/ 67.0	18	53.5/ 45.5	6	3.0/ 3.0	**293.2/ 252.0**
Architectures 2,4	31	120.8/ 103.2	35	77.0/ 67.0	18	53.5/ 45.5	1	0.5/ 0.5	**251.7/ 216.2**
Architecture 3	26	101.3/ 86.6	30	66.0/ 57.5	18	53.5/ 45.5	2	1.0/ 1.0	**221.8/ 190.5**
Architecture 5	26	101.3/ 86.6	27	59.4/ 51.7	6	17.8/ 15.2	20	10.0/ 10.0	**188.5/ 163.4**

Table 6.4 Length of four wire types and total lengths of wires in different vehicle electronic architectures

6.1.2.3 The Estimated Total CAN Wire Length in Different Electronic Architectures

The lengths of CAN wires can also be estimated in a similar way. The CAN bus is wired along the side of the vehicle and the ECUs are linked to the bus at their nearest points along the line. The CAN bus lies along the side of the vehicle. In all the architectures, except the *Architecture 5,* there is no ECU towards the rear. Hence the CAN bus only lies between the mid and front of the vehicle. Its length is, therefore, equal to the length of centre-wheelarch zone wire minus 0.5W (since CAN bus lies on the vehicle side, the distance between the centre zone and vehicle side of 0.5W is taken out). From the above approximation, the length of the CAN bus of all the architectures except *Architecture 5,* is estimated at 2.04 and 1.72 metres for the Mercedes and Vauxhall, respectively.

The *Architecture 5* CAN bus needs to cover the front to rear wheelarch distance. Its length is, therefore, twice that of the other architectures. Its length is estimated at 4.09 and 3.44 metres for the Mercedes and Vauxhall, respectively.

All the ECUs are connected to the CAN bus with half a vehicle width length wire. Note that two distributed wheel ECUs are one vehicle width away from the CAN bus, while the other two are virtually next to the bus, so the average distance is from ECU to the bus is half a vehicle width. Table 6.5 shows the total CAN wiring lengths of all the architectures except *Architecture 1.*

Also there is a different in CAN wiring length between *Architectures 2* and *4,* where there is none before. This is because *Architecture 4* requires data integrated control between 4WS ECU and other ECUs, hence 4WS ECU needs to be on CAN. On the other hand, 4WS ECU requires no integrated control in *Architecture 2.*

Vehicle Electronic Architectures	**Length of CAN Wires (m) (Mercedes Benz S-Class / Vauxhall Corsa)**		
	CAN bus length	**ECU Wire Links to CAN bus length**	**Total CAN wiring length**
Architecture 2	2.04/1.72	4.64/4.03	6.68/5.75
Architecture 3	2.04/1.72	1.86/1.61	3.90/3.33
Architecture 4	2.04/1.72	5.57/4.83	7.61/6.55
Architecture 5	4.09/3.44	5.57/4.83	9.65/8.27

Table 6.5 CAN wiring lengths of the four vehicle electronic architectures

6.1.2.4 The Estimated Weight of Wiring Harness in Different Electronic Architectures

The weight of a thin wall automotive wire of conductor size 0.65 mm^2, made by MULTICOMP, which is comparable to the 0.5 mm^2 wires used in Rover 800 is used to estimate the total wiring weight of each vehicle electronic architecture [17]. The 500 metre of this wire weighs 5.85 kilograms, which is equivalent to 1.17kilograms/100metres. A twisted pair wire, which is used as networking bus in automotive applications [20-21], is to be used to estimate the CAN wiring weight. The wire, made by Belden Wire & Cable, weighs approximately 5.92kg/100m [19]. Table 6.6 displays the total number of wires, their total lengths, and weights of different vehicle electronic architectures.

Vehicle Electronic Architectures	**Wire Lengths (m) and Weights (kg) (Mercedes Benz S-class / Vauxhall Corsa)**				
	Total number of wires	**Wire length (m)**	**Wire weight (kg)**	**CAN wire weight (kg)**	**System wire weight (kg)**
Architecture 1	100	293.2/252.0	3.43/2.95	0/0	3.43/2.95
Architecture 2	85	251.7/216.2	2.95/2.53	0.40/0.34	3.34/2.87
Architecture 3	76	221.8/190.5	2.59/2.23	0.23/0.20	2.83/2.43
Architecture 4	85	251.7/216.2	2.95/2.53	0.45/0.39	3.40/2.92
Architecture 5	79	188.5/163.4	2.21/1.91	0.57/0.49	2.78/2.40

Table 6.6 Total number, lengths and weights of wires in different vehicle electronic architectures

Compared to *Architecture 1*, *Architectures 2, 3, 4, and 5* enable weight saving of 3%, 17%, 1% and 23% on a large passenger car respectively. The four architectures, when installed in small vehicle, would yield 3%, 18%, 1% and 19% in weight saving from the *Architecture 1*, respectively.

The results of the above weight prediction suggest that the relative wiring weight of all the electronic architectures is consistent across small and large vehicle range.

6.2 Cost Comparison Between Different Architectures Wiring Harness

Cost is a very important aspect in car manufacturers' considerations, since the automotive market is very competitive. Before deciding to include or change to a new system, a car manufacturer will contemplate system cost among the first few factors. In this section, the wiring harness cost and its effects on cost of different vehicle electronic architectures, will now be discussed. It should be note that not all the costs are quantifiable since the study is in equivalent of a concept stage. An attempt was, however, made to estimate the component costs of different architectures. The costs of wiring, microcontrollers, peripheral ECU components, sensors and actuators were estimated or collected. These add up to system component cost. The results, together with the discussion on other cost aspects, can provide a comparative indication of the potential costs and savings of different electronic architectures. The future trend of costs will also be discussed.

6.2.1 Component Cost Estimation

6.2.1.1 Wiring Harness Cost

From [22], the cost of wires when fitted into a vehicle depends primarily on the number of wires, and not their lengths. This is due to the fact that most of the cost is incurred from preparation, such as wire cutting and peeling. The total materials and connecting cost of system wiring harness can be approximated from the number of wires (or signals) in a system.

The cost per wire was 30p in 1993 [22]. Since most of the cost if from preparation, it is assumed to rise with inflation, as does labour cost. Assuming the inflation rate of 2.5% annually, the cost per wire in 2000 is predicted to be 36p.

The cost of CAN installation is also included here due to its relation to wiring. From [22], the cost of adding CAN to an ECU can be coarsely approximated by equation 6.1.

CAN cost per ECU = (CAN + INTERF) x 1.8 + 2WIRE (6.1)

CAN	- CAN chip cost
INTERF	- CAN interface material cost (e.g. drivers, crystal, etc. ≈ 0.61)
1.8	- on cost overhead multiplier (e.g. labour, PCB accessories, etc.)
2WIRE	- twisted pair serial bus (60p)

The cost variation of all the factors except CAN chip, from model year 1994 vehicles estimated in [22], is not known. They are thus still used in this estimation. The cost of CAN chip is currently £1.12 [23]. The CAN installation cost per ECU in 2000 is estimated at:

(1.12 + 0.61) x 1.8 + 0.60 = £3.71

Using the wiring and CAN installation cost estimation described above, the wiring harness costs of all the architectures are shown in Table 6.7.

Vehicle electronic architectures	Number of wires	Wiring costs (£)	No. of CAN capable ECU	CAN installation costs (£)	Wiring harness costs (£)
Architecture 1	100	36	0	0	36
Architecture 2	85	30.6	5	18.6	49.2
Architecture 3	76	27.4	2	7.4	34.8
Architecture 4	85	30.6	7	26.0	56.6
Architecture 5	79	28.4	6	22.3	50.7

Table 6.7 Wiring harness costs of different electronic architectures

6.2.1.2 Microcontroller Costs

The microcontroller costs are obtained, based on the market price of the microcontrollers specified in the performance prediction in Chapter 5. The prices of microcontrollers of the same or nearest family (in number of bits) as the ones specified, with the closest amount of ROM and RAM are taken as the basis for calculation. It is estimated in [33] that 1K of ROM costs approximately $0.05, whereas the same amount of RAM costs 10-15 times more. 1K of RAM would cost $0.75 in the worst case approximation.

The memory difference between the specifications and the available microcontrollers is then estimated into costs. This is applicable in industry when a Original Equipment Manufacturer (OEM) wants to buy a large number of microcontrollers, a manufacturer will fabricate it according to the customer's specifications. The ROM/RAM additional or reductive costs are converted into pounds, assuming $1.4 to £1 exchange rate. These costs are added or subtracted to the microcontroller price to complete the microcontroller cost estimation. The results are displayed in Table 6.8.

Microcontroller Cost Estimation	***Architectures 1, 2, 4***			***Architecture 3***	***Architecture 5***	
	Active suspension	*4WS*	*ABS*	*Total centralised ECU*	*Central ECU*	*Distributed wheel ECU*
Required Microcontroller	MC68HC08	MC68HC05	MC68HC08	MC68HC12	MC68HC05	MC68HC08
Required ROM (K bytes)	**7.6**	**5.0**	**7.1**	**17.2**	**10.4**	**7.8**
Required RAM (K bytes)	**0.6**	**0.4**	**0.5**	**1.3**	**0.8**	**0.6**
Based microcontroller price (£)	13.0*[34]	9.3[35]	13.0	24.9[36]	9.3	13.0
Required – available ROM	-24.4	-11.0	-24.9	22.3	-5.6	-24.2
Required – available RAM	0.1	-0.1	0.0	0.3	0.3	0.1
Estimated MC cost (£)	**12.1**	**8.8**	**12.1**	**25.9**	**9.2**	**12.2**

Table 6.8 Estimated microcontroller costs

* all the prices of microcontrollers and electronic components are quoted per piece. This would certainly be more expensive than the actual production prices, which have bulk discounts. They are, however, useful for comparison purpose.

6.2.1.3 Peripheral ECU Component Costs

The costs of other ECU components namely peripheral Integrated Circuits (IC), resistors, capacitors and transistors, are also added to the ECU cost. There is no formula in estimating these component costs during the design stage. It is, however, believed that their number may be related to the number of ECU inputs and outputs. This is because they are primarily to assist a microcontroller in processing the inputs and outputs. The estimation was carried out by plotting a graph of the number of input and output signals of drivetrain or dynamic control ECUs produced from 1986 to 1995 (the number of signals were simply the number of pins they possessed), and the number of their electronic components, to obtain the 1st order trend line. Figure 6.9 displays the graph produced.

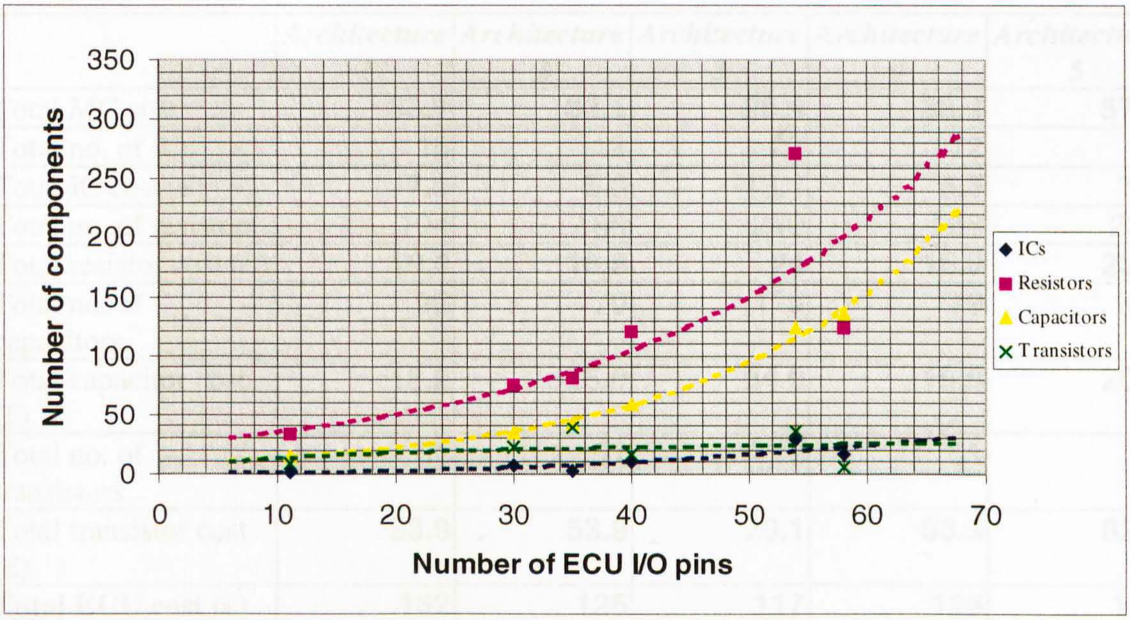


Figure 6.9 Plot of number of ECU components and I/O pins

All the ECUs studied in the graph were manufactured using through hole technology. It is acknowledged that recent changes to semiconductor device packaging (surface mount) may affect this assumption. Another factor, which should affect the trend lines, are the unknown number of unused pins in each ECU.

By applying the number of input and output signals from the wiring models of *Architectures 1-5*, displayed in Figures 6.2 to 6.5, to the trend line, the number of vehicle drivetrain and dynamic control system components could be predicted.

The costs of the peripheral ICs, resistors, capacitors and transistors were taken from [17], and averaged. The products of the average component prices and their predicted numbers are then used to form ECU component costs. The results are shown in Table 6.9. The details of calculation are shown in Appendix D.

	Architecture 1	*Architecture 2*	*Architecture 3*	*Architecture 4*	*Architecture 5*
Total MC cost	**33.1**	**33.1**	**25.9**	**33.1**	**57.9**
Total no. of ICs	15	12	27	12	15
Total IC cost (£)	**7.3**	**5.8**	**13.1**	**5.8**	**7.3**
Total no. of resistors	194	168	240	168	232
Total resistor cost (£)	**19.4**	**16.8**	**24**	**16.8**	**23.2**
Total no. of capacitors	96	79	179	79	108
Total capacitor cost (£)	**18.2**	**15.0**	**34.0**	**15.0**	**20.5**
Total no. of transistors	51	51	19	51	78
Total transistor cost (£)	**53.9**	**53.9**	**20.1**	**53.9**	**82.4**
Total ECU cost (£)	**132**	**125**	**117**	**125**	**191**

Table 6.9 Microcontrollers and peripheral ECU component costs

6.2.1.4 Sensor and Actuator Costs

The costs of sensors and actuators can be calculated from the combined number of control related sensors and actuators of each system. Individual sensor and actuator prices are taken from [17,19]. The sensor and actuator types are the ones used in their corresponding system found in literature [7]. Table 6.10 shows their combined costs.

	Architectures 1, 2	*Architectures 3, 4, 5*
No. of microswitches	6	6
Total microswitch cost (£)	**5.8**	**5.8**
No. of steering angle sensors	1	1
Total steering angle sensor cost (£)	**14.2**	**14.2**
No. of speed sensors	6	5
Total speed sensor cost (£)	**38.4**	**32**
No. of acceleration sensors	7	7
Total acceleration sensor cost (£)	**175.3**	**175.3**
No. of height sensors	4	4
Total height sensor cost (£)	**22.8**	**22.8**
Total sensor cost (£)	**250**	**243.6**
No. of solenoid actuators	8	8
Total actuator cost (£)	**75.4**	**75.4**

Table 6.10 Combined sensor and actuator costs

6.2.1.5 Overall System Costs

The costs of all the system components can now be summed up to obtain the total system costs. This is shown in Table 6.11.

	Architecture 1	*Architecture 2*	*Architecture 3*	*Architecture 4*	*Architecture 5*
Wiring cost	36	49.2	34.8	56.6	50.7
ECU cost	133.9	126.6	117.9	126.6	195.1
Sensor cost	250.0	250.0	243.6	243.6	243.6
Actuator cost	75.4	75.4	75.4	75.4	75.4
Overall system cost	**493**	**499**	**471**	**500**	**561**

Table 6.11 Overall system costs

The pie graphs of all the architecture costs, showing how much each component is contributed towards overall costs, are displayed in Figures 6.10.

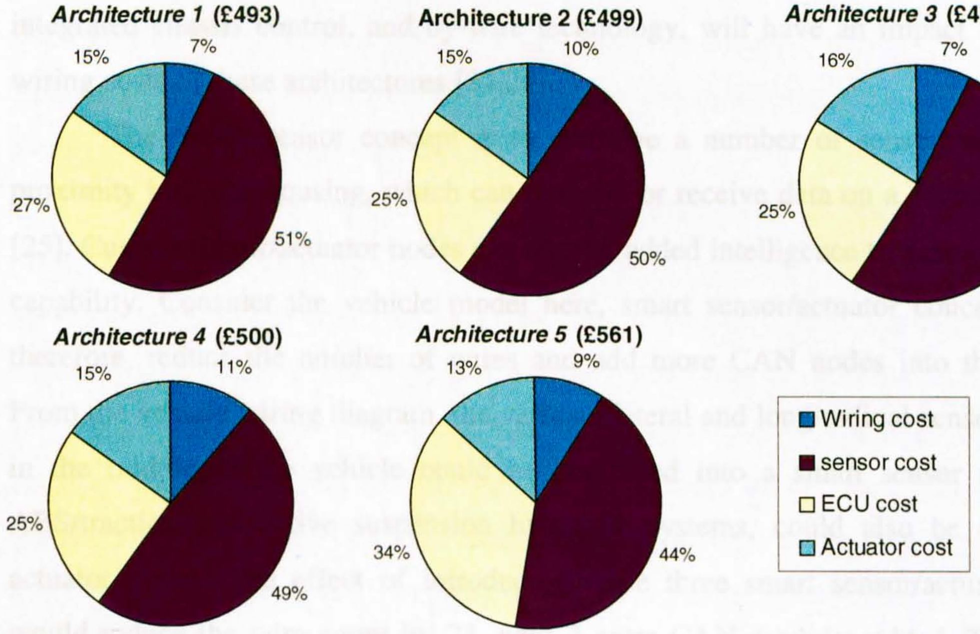


Figure 6.10 System costs of different architectures

6.2.2 Analysis

The overall system costs indicate a cost advantage on *Architecture 3*, primarily due to having only one ECU. The costliest *Architecture 5* is approximately 19% more expensive, while the other three architectures have comparable component costs, being around 6% more expensive than *Architecture 3*.

The sensor costs dominate the overall system costs, accounting roughly for half of the costs in all cases. The wiring costs have the least impact of all, contributing not more than 11% to the total costs. Since the sensor and actuators costs are almost consistent across all architectures, the ECU prices hold the key to overall system costs. *Architecture 5*, which has the most number of ECUs, is therefore the most expensive.

Apart from the component costs, other factors and future trends affecting the overall system costs will now be discussed.

6.3 Future Trends and Other Effects on System Costs

6.3.1 Future Trend Effects

In the near future, the imminent introduction of smart sensors/actuators, integrated chassis control, and by-wire technology, will have an impact on vehicle wiring costs of these architectures [24,26].

The Smart sensor concept is to combine a number of sensors/actuators in proximity into one housing, which can transmit or receive data on a vehicle network [25]. Current sensor/actuator nodes can also be added intelligence to have networking capability. Consider the vehicle model here, smart sensor/actuator concept would, therefore, reduce the number of wires and add more CAN nodes into the vehicle. From the vehicle wiring diagram, the vertical, lateral and longitudinal sensors located in the middle of the vehicle could be combined into a smart sensor node. The ABS/traction and active suspension hydraulic systems, could also be two smart actuator nodes. The effect of introducing these three smart sensor/actuator nodes would reduce the wire count by 23, with 3 extra CAN modules added. This would result in cheaper wiring cost to implement the *Architectures 2-5*.

Integrated vehicle chassis control involves closer co-operation between different chassis control systems, by sharing data and functioning together during certain driving conditions to improve handling and comfort [27-28]. An example of this is the activation of active suspension during braking to prevent dive, to improve passenger comfort. More signal sharing between ECUs would greater utilise the bandwidth of CAN. The effect on the vehicle wiring architectures here, would include the introduction of 4WS ECU to CAN on *Architecture 2*, and the increase in wire count on *Architecture 1*, due to more signal sharing.

By-wire technology, most notably brake and steer by-wire, would use electro-mechanical brakes and electric steering actuators to replace hydraulic brakes and steering columns, respectively [29]. Though by-wire may take longer time before it can be implemented on vehicles than the two aforementioned technologies due to safety and car electrical power concern, its substantial potential benefits making its application imminent. The effect of by-wire technology to wiring would possibly be an increase in ECUs and electronic contents. Again this would increase CAN utility and for a vehicle with CAN, or raise the wire count in a vehicle without CAN.

The effects of these coming technologies combined would clearly substantiate the use of networking. Electronic architectures that include networking would render savings in terms of wiring cost over ones without networking. Consider the vehicle electronic architectures here, *Architectures 2-5* would have far cheaper wiring cost than *Architecture 1*. *Architecture 3* would have the cheapest wiring cost due to the smaller number of CAN nodes and slightly fewer signals, followed closely by *Architecture 5 and Architectures 2 and 4*.

The concept of integrating electronic contents with its physical components have been seriously considered, especially in powertrain systems. It is believed that by integrating the EMS to, for example, the intake manifold in the engine and the transmission control inside the transmission, yield benefits in material cost savings [1]. Some interconnections can be eliminated, while part of the EMS enclosure is provided by the manifold. *Architectures 1 and 2* could benefit from this concept by having their power steering ECUs integrated with their mechanical contents [30]. 4WS ECU could possibly be integrated with its hydraulic system also. *Architecture 5* could also benefit from these cost savings, by having its wheel ECUs integrated to wheel brake or suspension actuators.

6.3.2 ECU Development Cost

It is generally acknowledged that system complexity has significant effect on development cost and time. With this understanding, *Architecture 4* is expected to be more costly in development. This is because simultaneous development of the architecture requires close corroboration between the ABS, EMS, active suspension and 4WS suppliers. This would make the development time and cost of these systems higher than having them individually developed, as *Architectures 1 and 2*.

Architectures 3 and 5, though contain highly complex systems of engine & powertrain and vehicle dynamic control ECUs, could actually yield cost savings, due to the fact that they only require a couple of complex systems developed rather several fewer complex systems [26]. This, however, depends on the number of suppliers with capability to develop these complex systems. The more suppliers there are, the more competitive the market becomes, and hence cheaper costs. Complex systems also have a drawback in terms of the limited number of car models that they can be fitted in. Individual ABS, power steering or cruise control ECUs can be fitted to a wide range of models, whereas a more complex vehicle dynamic control ECU may have to be specific to only one or two car models. Therefore, this advantage of the economy of scale is towards *Architectures 1,2,4*.

6.3.3 Assembly and Diagnostic Costs

Assembly cost depends much on labour cost and hence how long it takes to assemble the systems. The wire count is one of the indicators in this case. The level of system integration is another, since highly integrated systems need less steps to be connected and assembled. These considerations favour *Architecture 3* due to its relatively small number of wires, and *Architecture 5* due to its potentially integrated system components.

Wiring is the significant source of electronic failures in cars [31]. Hence the less wiring a vehicle contains, the less likely it is to fail. The conventional centralised architecture would, therefore, have the potentially highest maintenance cost. Though ECUs are far less likely to fail, once fails it is almost always totally replaced by a new one, with high cost. The more complex the ECU, the more expensive it tends to cost. In this respect, *Architecture 3* could be more costly to maintain than the other four architectures.

6.4 Summary

The work done in this Chapter is to compare the five architectures in terms of component complexity and costs. The system components include sensors, actuators, ECUs and wiring.

The number of wires in each architecture was derived from published papers on the exemplary powertrain and dynamic control systems [5-12]. The results suggest that *Architecture 1* has the highest number of wires. *Architectures 2, 4* are expected to have 15% fewer wires, whereas the *Architectures 3 and 5* would have 24% and 21% fewer than *Architecture 1*, respectively.

The estimate of overall wire length, including that of CAN cables, was made based on the dimensions of small and large luxury vehicles in the market. From the wire lengths, the wiring weight of the architectures were calculated and compared. The results show that the percentage of weight different between architectures is almost consistent across the two vehicle types. *Architectures 2, 4* have a slight weight saving (approximately 2%) on *Architecture 1*, whereas *Architectures 3, 5* enable weight reduction of 20%.

The component costs of the alternative architectures were then calculated. Each wiring cost is considered fixed regardless of length, since the cost is mainly from preparation [22]. The costs of CAN, including cables, microcontrollers and interface materials were calculated from the equation in [22]. ECUs consist of microcontrollers and peripheral electronic components. Microcontroller specifications were estimated in Chapter 5, while the number of peripheral components was approximated from the plot of number of ECU I/O pins against peripheral components. Individual component price was taken from electronic supplier catalogues [17,19].

The overall system costs indicate that *Architecture 3* is potentially the cheapest system. *Architecture 1, 2, 4* costs are similar at 6% higher, while *Architecture 5* is potentially the most expensive being 20% costlier than *Architecture 3*. The future automotive electronic trends and other cost aspects that affect the above cost estimates were discussed.

6.5 References

1. *De Vos G, et al.* **Migration of Powertrain Electronics to On-Engine and On-Transmission** SAE No.1999-01-0159
2. *Anderson D* **A New Way of Trouble Shooting the Wire Harness from Drawing Board to Service Rack** SAE No.960394
3. *McLaughlin R, et al.* **A feasibility study of CAN technology in body electronic control systems** IMechE Autotech 95, 1995
4. *Bauer H* **Automotive Brake Systems** Bosch, 1995
5. *Matsutomi S, et al.* **Development of ABS and traction control computer** SAE No.901707
6. *Achleitner E. et al.* **Electronic Engine Control System for Gasoline engines for LEV and ULEV Standard** SAE No.950479
7. *Jurgen R* **Automotive Electronics Handbook** McGraw-Hill, 1995
8. *Bauer H* **Bosch Automotive Handbook 4th edition** Bosch, 1996
9. *Schleupen R, et al.* **Electronic Control Systems in Microhybrid Technology** SAE No.950431
10. *Sato H, et al.* **Development of Four Wheel Steering System Using Yaw Rate Feedback Control** SAE No.911922
11. *Ise K, et al.* **The 'Lexus' Traction Control (TRAC) System** SAE No.900212
12. *Graham C, et al.* **General Motor High Performance 4.3L V6 Engine** SAE No.920676
13. *Yokaya Y, et al.* **Integrated Control System Between Active Control Suspension and Four Wheel Steering for the 1989 Celica** SAE No.901748
14. *Irie N, et al.* **4WS technology and the prospects for improvement of vehicle dynamics** SAE No.901167
15. *Ting C* **A Novel Approach of Transmission Line Theory in EMC Assessment (PhD Thesis)** July 2000
16. **Car Magazine** Issue 418, June 1997
17. **RS Catalogue** RS Components Ltd., 2000
18. **Rover 800 Series Electrical Fault Finding Manual** Rover Group, 1992
19. **Farnell Component Catalogue** Farnell 2000
20. *Wheat G, et al.* **Vehicle Multiplex Wiring – An Implementation** SAE No.880591
21. *Akashi K, et al.* **Application of Multiplexing to Automotive Body Electrical Control** Fujikura Technical Review Issue 22, 1993
22. *McLaughlin R* **In-Vehicle Communication Networks (MSc Thesis)**, 1993
23. **Philips PCA82C250 CAN Controller Interface price list** RS Components catalogue website http://www.rswww.com, June 2000
24. *Ward D, et al.* **A Vision of the Future of Automotive Electronics** SAE No.2000-01-1358
25. *Sparks D, et al.* **Multi-Sensor Modules with Data Bus Communication Capability** SAE No.1999-01-1277
26. *Bannatyne R* **Electronic Braking Control Developments** Automotive Engineering International, Feb 1999
27. *Kawakami H, et al.* **Development of Integrated System Between Active Control Suspension, Active 4WS, TRC and ABS** SAE No.920271
28. *Yokoya Y, et al.* **Integrated Control System Between Active Control Suspension and Four Wheel Steering for the 1989 CELICA** SAE No.901748

29. *Jordan M* **Drive-by-Wire Will End the Era of the Hnadbrake Turn** Electronic Engineering, Dec 1999
30. *Burns J, et al.* **Integrated Motor Drive Unit A Mechatronics Packaging Concept for Automotive Electronics** SAE No.2000-01-0132
31. *Anderson D* **A new way of trouble shooting the wire harness - From drawing board to service rack** SAE No.960394
32. **Mercedes-Benz S600 dimensions** Mercedes-Benz UK website http://www.mercedes-benz.co.uk, June 2000
33. *Tindell K* **Embedded Systems in the Automotive Industry** Embedded Systems Conference, 1999
34. **Motorola MC68HC705L16CFU Microcontroller Price, Arrow Catalogue** Arrow Electronics UK, 2000
35. **Motorola MC68HC908GP32 Microcontroller Price,** Maplin, 2000
36. **Motorola MC68HC16Z1CFC16 Microcontroller Price, RS Catalogue** RS Components Ltd, 2000

CHAPTER 7

COMPONENT RELIABILITY PREDICTIONS OF ALTERNATIVE ELECTRONIC ARCHITECTURES

This chapter introduces the concept of reliability, presents alternative methods for its prediction, and selects and uses one of these to compare vehicle electronic architectures.

Reliability is defined as 'The ability of an entity to perform a required function under given conditions for a given time interval' [1].

As the trend in the increasing amount of electronics in automotive continues, the quality and reliability of electronics becomes prominent. Customers expect vehicle electronics to function correctly throughout the life of a vehicle [2]. Since electronics is now widely applied in the control of safety related systems of a vehicle such as Anti-lock Braking Systems (ABS) and power assisted steering systems, failures of these systems could lead to dangerous driving situations [3]. Hence careful consideration has to be placed upon its reliability from an early design stage. It is suggested by [4] that the engineering effort put into early part of the design of a product gives more effective results in terms of reliability, quality, cost and time, than the effort put in the later stage. This stresses the importance of reliability consideration in alternative architecture analysis.

During the initial design stage of a new vehicle system, before hardware is available, reliability prediction is useful to quantitatively evaluate whether the design will meet the target reliability level [5]. In some cases, the reliability prediction may not be very accurate, but it can identify a better design by comparison between a number of initial designs.

7.1 Types of Reliability Prediction Techniques

There are a number of reliability prediction techniques that have been developed. The designer can select one, which suits the depth of design knowledge and the historical data on equipment reliability available to him/her [5]. The reliability estimate is usually in terms of Mean Time Between Failure (MTBF) or Mean Time

To Failure (MTTF). MTBF is an average time that an equipment can be used before it fails, when the equipment is repairable [6]. MTTF is similar to MTBF except that it describes failure rate of non-repairable equipment. According to the data type or availability, the techniques can be classified into five categories, as described below [5].

7.1.1 Similar Equipment and Similar Complexity Techniques

These two are among of the most basic techniques, developed to perform a very early prediction of system reliability, before system specifications are available. The objectives of the predictions using these techniques are mostly to estimate if a newly introduced system will meet a minimum reliability constraints [5].

The predictions are based upon historical reliability data of existing equipment, which is similar in type, operation or complexity to the one being predicted. For instance, a historical reliability data of an airbag system accelerometer can be a reliability indicator of a set of similar accelerometers of a new ABS system. Reliability data of a current EMS ECU should provide a reasonable reliability source for a new version EMS ECU.

The accuracy of this technique depends on the quality of reliability data and similarity between the new equipment and the selected one, for which the field results are used. The more accurate prediction is expected if the production techniques used and the manufacturers of the two equipment are the same.

7.1.2 Prediction by Function Techniques

Statistical correlation between major functional characteristics and the history of operational reliability of an equipment is applied in this technique. Each characteristic is given a weighing factor according to its significance in terms of reliability. An example given by [5] shows that a radar reliability prediction can be calculated based on the derived correlation equation between average number of component parts and other radar characteristics. The characteristics include design year, detection range, target resolution, receiver dynamic range, etc. Having estimated the average number of component parts, the radar MTBF can then be obtained from the average failure rate per part.

Different prediction equations were derived for several equipment by [7]. No literature was found on the application of this technique on automotive electronics, but the technique is considered applicable to the field of automotive electronics [5].

7.1.3 Part Count Techniques

The part count techniques can be applied when information on the number of component parts and types is available or can be estimated. The basic prediction is done by first finding out the number of each type of components classified by the failure data source. The generic failure rate associated with each class of component is then multiplied by the number of components. All the component failure rates are then added up to make the equipment failure rate. The general governing equation is as follows [5]:

$$\lambda = \sum_{i=1}^{n} N_i (\lambda_G \pi_Q)_i \qquad (7.1)$$

for a given equipment environment where:

λ - total equipment failure rate (failures/10^6 hours)
λ_G - generic failure rate for the i^{th} generic part (failures/10^6 hours)
π_Q - factor affecting the reliability of the i^{th} generic part
N_i - quantity of i^{th} generic part
n - number of different generic part categories

There are several sources of generic failure rates, with the most well known one being [8] produced by the US government. References 9-11 are examples of other sources. Many large companies also compile their own database of component reliability based on their own field data.

7.1.4 Stress Analysis Techniques

This technique is similar to the previous one but takes into account the different levels of stress to which a component is subjected such as electrical load and vibration, and the environment it will be in such as temperature and humidity. These influencing factors are multiplied with the generic failure rate in the prediction model to reflect the effect of the stress or environment on the equipment. The model is therefore believed to give a more accurate prediction than the part count method.

An example of the model for silicon integrated circuits taken from [9] is shown in Equation 7.2.

$$\lambda = \lambda_B \, \pi_T \, \pi_Q \, \pi_E \tag{7.2}$$

Where

λ - the failure rate
λ_B - base(generic) failure rate
π_T - temperature weighting factor
π_Q - quality factor
π_E - application environment factor

Some of the sources of failure data such as [8, 9, 11] also provide data for the stress analysis technique with different types of stress factors included. Concerning the accuracy of these models, some work has been carried out to evaluate their accuracy and suggest an improved model. Whitehead, et al. [12] evaluated the accuracy of British Telecom (BT)'s models [9] on actual performance of telecommunication equipment, and concluded that the model accuracy was acceptable for first 1-3 years of equipment operation. Collas, et al. [13] compared the operational reliability results of mainframe computers with its predicted reliability model constructed using BT's models [9], and adjusted its model to improve accuracy. Kerscher, et al. [14] found that the Military handbook model [8] was not accurately representing the field reliability of its electronics equipment. They, therefore, developed their own prediction model based on the combined Military handbook model and the Weibull distribution technique.

7.2 Reliability Prediction of Automotive Electronics

It is pointed out by [5] that the most of the reliability prediction models could give questionable accuracy when predicting automotive electronic product reliability because of their assumption that the electronic component failure rate is constant. The practical electronic equipment have been found to have a time dependent failure rate which is represented by a bathtub curve as shown in Figure 7.1. This argument is supported by actual assessment results from [12, 14, 15, 16]. Furthermore, the suggestion that the assumption that component failure rate during non-operating period is zero, may cause an inaccuracy. This is because automotive electronic equipment could be in operation for only 5% of its lifetime [5]. Hence by omitting the non-operating failure rate, the model could give too optimistic a result.

As previously mentioned, the reliability prediction is not generally expected to give a precise value for such an initial design stage. It is, however, useful as a tool for estimating whether the new product reliability will meet preliminary reliability requirement, and for comparing the reliability of different designs.

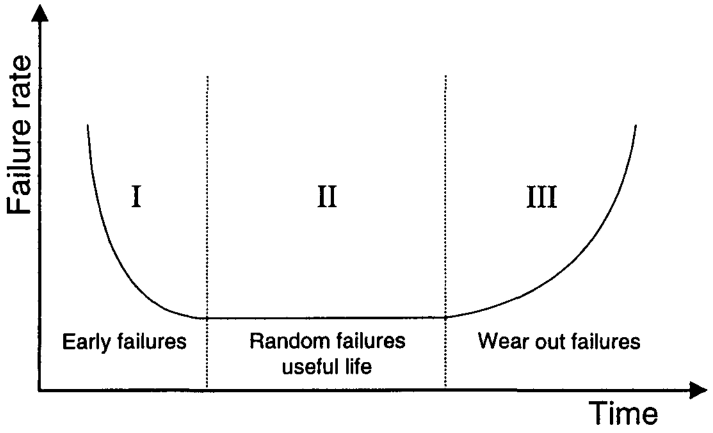


Figure 7.1 A bathtub curve representing typical component failure rate

7.3 Reliability Modelling of Different Vehicle Electronic Architectures

As mentioned above, a reliability prediction of electronics systems can provide initial comparison between systems, in the preliminary design stage before hardware is developed. This section is intended to compare the projected reliability of different electronic architectures.

In this case, only the Part Count and Stress Analysis techniques are applicable, due to the fact that they require no field data and not much information, other than a number of components in a system, to form a model. Furthermore, there is no universally accepted best model. Although the accuracy of this technique could not be proved, it is justifiable to apply it just to compare different architectures and find a more reliable one.

Of all the 4 reliability data sources available [8-11] for these techniques, [10] and [11] were chosen for the reliability modelling in this project. [10] was chosen as it is the most up-to-date. It was developed by AT&T following the guidelines of the US Ministry of Defence [8], which is among the most widely known electronic equipment reliability data sources. [11] was selected since, despite its relatively old information, it uses historical reliability data of components in automotive applications. By using two models, their results should provide a more substantial basis for architecture reliability comparison than applying only one method.

The following sections describe the two modelling processes.

7.3.1 Reliability Modelling Based on AT&T Technique

7.3.1.1 Modelling Technique

AT&T Bell Laboratories provide the estimates of hazard rates of the components used in products manufactured by AT&T [10]. The information is the basis for this reliability prediction method.

ECUs are considered to be 'nonmaintained systems' in the area of reliability engineering, since none of their electronic components are inspected and maintained during their operating lives. They are simply used until becoming faulty and hence being taken out for replacement or repair. The reliability of such system is commonly measured by hazard rate [10]. It is the instantaneous rate of failure for units of a population that last to time t.

At time t, let the number of units which work be S(t).

After a period of Δt, the number of units which still work is S(t+Δt).

Hence the number of units that fail during that period is S(t) - S(t+Δt)

The average hazard rate over Δt, Λ(t), is the fraction of failed units over the working ones. It can be represented by the equation:

$$\Lambda(t) = \frac{S(t) - S(t+\Delta t)}{S(t).\Delta t}$$

When Δt is approaching zero, the hazard rate is:

$$\lambda(t) = \lim_{\Delta t \to 0} \Lambda(t)$$

This eventually gives:

$$\lambda(t) = -\frac{d}{dt}\ln[S(t)]$$

$$S(t) = e^{-\int_0^t \lambda(t)dt} = e^{-\lambda t}$$

The unit of hazard rate is time^{-1} and is usually represented by FIT (Failure In Time). A device with a hazard rate of 1 FIT can be interpreted as it has the probability of 10^{-9} to fail in the next hour, given that it has survived up to that hour.

For example, an electronic system which has a hazard of 1000 FITs has a probability to work for 2000 hours without failure S(t):

$$\lambda(t) = 1000 \text{ x } 10^{-9} \text{ / hour}$$

$$S(t) = e^{-0.000001(2000)} = e^{-0.002} = 0.998$$

Hence the smaller the hazard rate value it has, the more reliable the device is.

The hazard rate model of [10] for electronic components is formed using two distinct sources of device reliability information: accelerated life tests and performance recorded in the field or factory. The result is a model, which is a combination of a Weibull and an exponential distribution, representing short and long term hazard rates, respectively. Figure 7.2 illustrates the model in log scale.

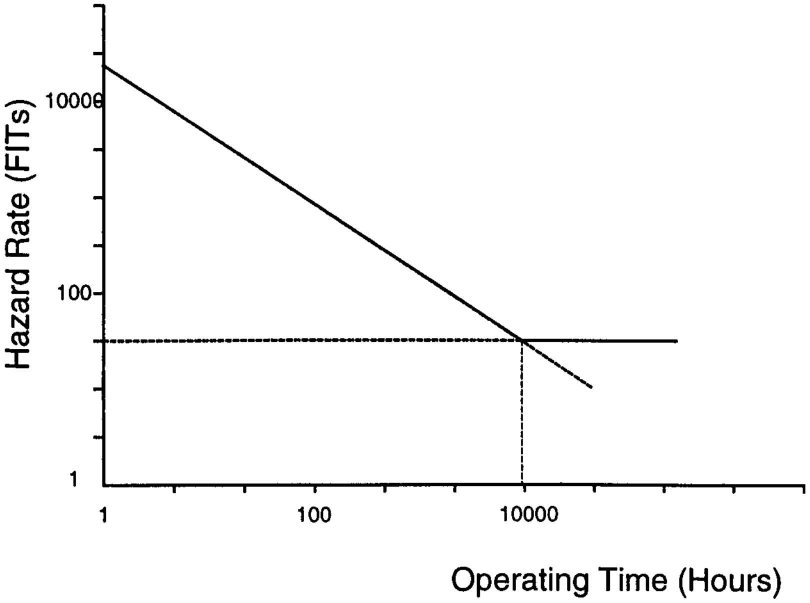


Figure 7.2 Hazard rate model developed by AT&T [10]

For an automotive electronic system, its reliability over the vehicle life span, considered long-term (or steady-state) reliability, is of interested. To form a reliability model, the number of different system components are multiplied by their associated exponential device hazard rates and their contributing factors. The governing equation is shown in equation 7.3.

$$\lambda_{total} = E \sum_{i=1}^{all\ devices} (A_T)_i (A_E)_i (\lambda_L)_i \qquad (7.3)$$

Where:

λ_{total} - total hazard rate
E - environmental application factor
A_T - temperature acceleration factor
A_E - electrical stress acceleration factor
λ_L - device long-term hazard rate

Environmental application factor is related to where the ECUs are located, which is classified as vehicular-mounted in this case. The operating temperature is assumed within 40°C, the same in all the ECUs. Since all of the ECUs, except the distributed wheel ECUs of *Architecture 5* which are near wheelarch area, are assume to be located in a passenger compartment, the consistency of the assumption ensures fair comparison between the architectures. Electrical stress is applicable to passive components. The factor depends on power dissipated for resistors and voltage for capacitors. There is no information on these values, so they are assumed not under undue stress, corresponding to the electrical stress acceleration factor of 1.

7.3.1.2 Alternative Electronic Architecture Reliability Modelling

Here the application of the reliability modelling method described above is applied to the alternative electronic architectures.

As practised in Chapter 6, as the number of components in the Wheel Controller system was unknown, it had to be estimated. The estimation was carried out by plotting a graph of the number of input and output signals of known drivetrain and dynamic control ECUs produced from 1986 to 1995 (the number of signals were simply the number of pins they possessed), and the number of their electronic components, to obtain the trend line. Figure 7.3 displays the plot.

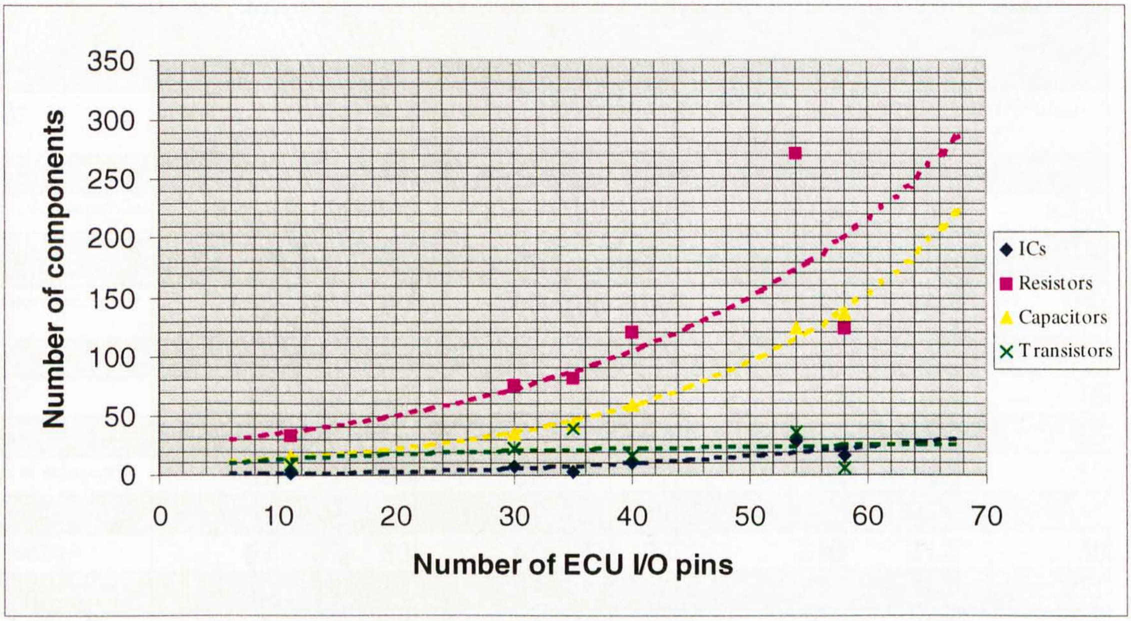


Figure 7.3 Projected number of peripheral components per ECU pins

By applying the number of input and output signals from the wiring models of the conventional and distributed architectures, displayed in Figures 6.2 and 6.5 of Chapter 6, to the trend line, the number of vehicle drivetrain system components could be predicted. It is acknowledged however, that recent changes to semiconductor device packaging (surface mount) may affect this assumption.

Applying the number of components to equation 7.3, the reliability predictions of all the alternative architectures are shown in Table 7.1. Note that the hazard rate of the majority type of a component is used as a representative for all types of that component. For example, most resistors found on the ECUs are of metal thin film type, so the hazard rate of the metal thin film resistor is used in the calculation of all the resistors. In the calculation, it is assumed that the components are connected in series, so the failure of one component means a system failure. This is not strictly true, but it is reasonable to consider that a component failure will affect the normal system operation.

	Architecture 1			**Architectures 2,4**			**Architecture 3**	**Architecture 5**	
ECUs	*Active suspension*	*4WS*	*ABS*	*Active suspension*	*4WS*	*ABS*	*Total centralised*	*Central ECU*	*Distributed ECU*
Number of I/Os	28	7	38	27	7	30	63	42	6
Type of microcontroller	8-bit	8-bit	8-bit	8-bit	8-bit	8-bit	16-bit	8-bit	8-bit
Hazard rate (λ_L) for microcontroller	100	100	100	100	100	100	100	100	100
Total microcontroller λ	**100**	**100**	**100**	**100**	**100**	**100**	**100**	**100**	**100**
Number of ICs	5	1	9	5	1	6	27	11	1
λ_L for ICs*	15	15	15	15	15	15	15	15	15
Total IC λ	**75**	**15**	**135**	**75**	**15**	**90**	**405**	**165**	**15**
Number of resistors	67	31	96	65	31	72	240	112	30
λ_L for resistors**	1	1	1	1	1	1	1	1	1
Total resistor λ	**67**	**32**	**32**	**65**	**31**	**72**	**240**	**112**	**30**
Number of capacitors	32	12	52	31	12	36	179	64	11
λ_L for capacitors***	0.2	0.2	0.2	0.2	0.2	0.2	0.2	0.2	0.2
Total capacitor λ	**6.4**	**2.4**	**10.4**	**6.2**	**2.4**	**7.2**	**35.8**	**12.8**	**2.2**
Number of transistors	17	16	18	17	16	18	19	18	15
λ for transistors****	20	20	20	20	20	20	20	20	20
Total transistor λ	**340**	**320**	**360**	**340**	**320**	**360**	**380**	**360**	**300**
Sum of component λ	**588**	**469**	**637**	**586**	**468**	**629**	**1161**	**750**	**447**
Environmental application factor (E)	8	8	8	8	8	8	8	8	8
ECU λ	**4707**	**3755**	**5099**	**4690**	**3747**	**5034**	**9286**	**5998**	**3578**
System λ	**13562**			**13470**			**9286**	**20309*******	
Probability of operating for a vehicle lifetime without failure******	**0.888**			**0.889**			**0.922**	**0.837**	

*digital CMOS 1-50 gates

**metal (up to 1MΩ)

***ceramic (general purpose) up to 0.1μF

****NPN or PNP

*****sum of a Centralised ECU λ and four Distributed ECU λ

******assuming vehicle lifetime of 20 years

Table 7.1 Reliability modelling of alternative architectures using AT&T data

As seen from the table, the reliability calculation suggests that *Architecture 3* is the most reliable system. *Architectures 1,2,4* and *Architecture 5* are predicted to be 4% and 9% less reliable respectively i.e. they should have 4% and 9% more probability of failure during operating lifetime (estimated 20 years) than *Architecture 3*.

7.3.2 Reliability Modelling Based on Automotive Reliability Data

7.3.2.1 Modelling Technique

This reliability technique is described in [11]. Its prediction is based entirely from a database of empirical automotive failure rates. The data was collected from a field failure data from a number of automotive sources, by an IIT Research Institute.

The prediction technique itself was developed further from earlier models [17,18], which is governed by the general equation 7.1. The influencing factors include component location in vehicle, component package type, module package, screen level, and burn-in, depending on the components. The predicting equations for specific types of component are shown in Appendix E.

The improvement over the earlier models was achieved by taking into account the effect of temperature, declining failure rate from infant mortality to steady-state operation, and nonoperating failure rate.

The failure rate given is in terms of failures per 100 part per 400 hours, as opposed to the hazard of the previous method. This is approximately for a year use of vehicle, assuming that it typically operates less than 5% of the time [11].

7.3.2.2 Alternative Electronic Architecture Reliability Modelling

Using the estimates of ECU components from number of ECU I/Os against component plot in Figure 7.3, described in the previous technique, the system reliability prediction was made. The details calculation is shown in Appendix E. The results are shown in Table 7.2.

	Architecture 1	*Architecture 2,4*	*Architecture 3*	*Architecture 5*
Failure rate	0.39	0.37	0.33	0.59
Failure rate (x with steady-state multiplying factor)*	0.21	0.20	0.18	0.32
Probability of failure in vehicle lifetime operation**	0.043	0.040	0.036	0.064
Probability of operating for a vehicle lifetime without failure	**0.957**	**0.960**	**0.964**	**0.936**

* for a typical automotive module over 30,001 hours of operation [11]
** assume vehicle life of 20 years.

Table 7.2 Reliability modelling of alternative architectures using automotive reliability data

The prediction indicates that *Architecture 3* is potentially the most reliable architecture. *Architectures 1,2,4, and Architecture 5* are predicted to be 1% and 3% less reliable, respectively.

7.3.3 Analysis

The results from both the reliability prediction techniques agree on the relative reliability of alternative architectures. Both suggest that the *Architecture 3* is potentially the most reliable, followed by the comparable *Architectures 1,2,4,* whereas *Architecture 5* is the least reliable. This could be explained by the fact that *Architecture 5* contains more ECUs than *Architectures 1,2,4,* which have the same number of ECUs. *Architecture 3* benefits most from having a single ECU.

The difference of the predicted failure rates between the two techniques underlines the caution, stated earlier, that the prediction at this early stage is unlikely to give an accurate result. It merely indicates a potentially more reliable design from others. Nonetheless, both the prediction techniques suggest that all the alternative architectures are highly reliable, with less than 15% chance of failure during a vehicle lifetime. This is slightly optimistic due to the fact that the reliability prediction of diodes is excluded from the models, because from the ECUs studied, the number of diodes and ECU pins are virtually unrelated. This should not strongly undermine the prediction results since the diode base failure rate is comparable to that of capacitors (in [11], while not all types of diode data is available in [10]), where there are noticeably fewer number.

In the view of overall vehicle component reliability, *Architecture 1* powertrain ECUs lack the benefit of component redundancy as a result of networking, which the other four architectures possess. This can be seen from them having the most number of wires, and hence more I/Os and components, from the wiring diagrams (Figure 6.2-6.5) in Chapter 6. *Architectures 3 and 5* gain further benefit from having integrated powertrain control, which reduces its I/Os resulting in having fewer components, and hence higher reliability. The powertrain ECU reliability was not included in the reliability prediction, because not all of the signal and I/O information on an EMS is available. The wiring diagrams in Chapter 6 were drawn by omitting a large number of signals between the engine and EMS, treating them as internal. Those signals and I/Os are essential for EMS component estimation, and hence for reliability prediction.

The advantage of higher reliability of *Architecture 3* may not be significant in terms of cost to customers. Since ECU failure is generally fixed by replacing it with a new one. Due to its higher complexity, the cost of its single ECU would be much more than any ECU of other architectures. This makes its replacement much more costly, despite a lower chance of failure.

7.4 References

1. *International Electrotechnical Commission* **International Vocabulary Dependability and Quality of Service** IEC 50 (191), 1991
2. *Rose P* **Automotive and aerospace electronic systems. Dependability requirements** Microelectronics and Reliability Vol.36 Iss.11-12 October, 1996
3. *Idoguchi M* **A Method of Reliability Analysis for Automotive Electronic Systems** SAE No.910355
4. *Kerscher W* **Reliability Prediction Techniques - Electrical/Electronic Products** SAE No.870051
5. *Electronic Reliability Subcommittee* **Automotive Electronics Reliability Handbook** SAE 1987
6. *Leitch R* **Reliability Analysis for Engineers An Introduction** Oxford University Press 1995
7. *James L, et al.* **Study of Reliability Prediction Techniques for Conceptual Phases of Development** RADC-TR-74-235, October 1974
8. **Reliability Prediction of Electronic Equipment** US Mil-Hdbk-217D, January 1982
9. **Handbook of Reliability Data for Components Used in Telecommunications Systems** British Telecom, Issue 4, January 1987
10. *Klinger D, et al.* **AT&T Reliability Manual** Van Nostrand Reinhold, 1990
11. *Denson W, et al.* **Automotive Electronic Reliability Prediction** SAE No.870050
12. *Whitehead A, et al.* **Reliability Performance of Electronic Components; A Critical Appraisal of British Telocomm's HRD Issue 4** 1987
13. *Collas G, et al.* **Comparison Between Operational and Predicted Reliability for Computer System: Modelling Adjustment** International Conference on Reliability Techniques and Their Application, 1991
14. *Kerscher W* **Failure-Time Distribution of Electronic Components** Proceedings of the Annual Reliability and Maintainability Symposium, 1988
15. *Franklin D* **Component Selection Impact on Reliability** Proceedings to the WESCON 1996
16. *Brombacher A, et al.* **Simulation, A Tool for Designing-In Reliability** Quality and Reliability Engineering International Vol. 9, 1993
17. *Binroth W, et al.* **Development of Reliability Prediction Models for Electronic Components in Automotive Applications** SAE No.840486
18. *Coit D, et al.* **Impact of Nonoperating Periods on Equipment Reliability** RADC-TR-85-91, September 1984

CHAPTER 8

CONCLUSIONS

The work in this thesis was initiated from the increasing trend in installing electronic control systems in vehicles. The interaction and interconnection of these systems is non-trivial and offers potential benefits in functionality, cost, weight and reliability.

This project aims to compare automotive architectures for dynamic control electronics, in the light of the trend mentioned above. The system-level evaluation of different architectures was made with a specific purpose of evaluating the Distributed Wheel Architecture (*Architecture 5*) which has been proposed. It was evaluated along with the traditional Conventional Centralised (*Architecture 1*), the recently adopted and increasingly more popular Conventional Centralised with Limited CAN Interaction (*Architecture* 2), and Total Centralised (*Architecture 3*) and Conventional Centralised with Functional Integration (*Architecture 4*), both of which could potentially be fitted to vehicles of the near future.

The work done in the project can be summarised as follows:

- Simulation of dynamic control systems operation using data generated from the road test of an instrumented vehicle.
- Derivation of functional specifications of the electronic control systems of the alternative architectures. This was followed by the estimation of microcontroller specifications needed for the control tasks.
- Simulation of CAN message transfer of alternative electronic architectures within a vehicle to predict the CAN associated delay.
- Calculation of a projected component cost of the alternative electronic architectures.
- Reliability modelling and comparison of the alternative architectures.
- Evaluation of the viability of the distributed wheel architecture on the above terms.

These six areas will now be examined in turn.

8.1 Simulation of Control Systems

The simulation of dynamic control system operation was carried out to verify the operation of the exemplary ABS, 4WS and active suspension systems, which were to be applied throughout the project. This simulation gave the following conclusions:

- Since the road test data obtained did not include all of the input signals needed for the simulation, those missing inputs had to be synthesised from the available data.
- Vehicle speed data from road test and test track information, and the synthesised data are sufficient for the verification of directional response of the dynamic control system functions intended.
- Synthesising data in this case, however, is not suitable for detailed simulation. The simulation carried out here only shows how the control systems respond to varying inputs direction wise, but not in details with magnitude of control parameters shown. For example, when front wheel height inputs go below zero indicating diving, the simulation shows that the active suspension responds by sending positive going front wheel control signals. This shows satisfactorily the operation of active suspension, where the amplitude of control response is of no interest.
- To perform a quantitative simulation, more realistic synthesised data is needed. This requires information such as test vehicle spring and damper characteristics and test track surface angle, to generate wheel height data. However, as discussed in the wheel speed data generations, there are many influencing factors to each data. These make synthesised data inappropriate for high precision simulation, such as for control parameter calibration, owing to the additive nature of errors from each contributory input.
- The simulation in this thesis, however, benefits from using synthesised data, as it could be modified or exaggerated to demonstrate particular control functions, such as the ABS operation when a wheel is locking.
- Having verified the operations of the exemplary ABS, 4WS and active suspension controls, they were then to be used for the microcontroller specification estimation.

8.2 Microcontroller Specification

Microcontroller specification, including performance, ROM and RAM capacity and number of I/O, for each of the ECU in the alternative architectures was to be estimated, as part of the feasibility study for architecture implementation. The conclusions on the estimation of microcontroller specifications are as follows:

- Prediction of a microcontroller response time, of a control system at this equivalent of early design stage, is acknowledged to be difficult and potentially inaccurate. The prediction, based on past programming experience and software information on older version of similar systems, believed to be the only plausible method. This is the method currently adopted by ECU developers. Due to the lack of these factors, an alternative prediction, by constructing a pseudo code based on functional specifications, was developed here
- The prediction results of the two known ECUs were verified against the experimentally measured response time. The results were found to vary in accuracy, with the more detailed specification producing greater accuracy. This was mainly due to designer dependent program parameters, such as switch debounce and diagnostics. A 'multiplying factor' was hence calculated to assist the response time prediction of alternative architecture ECUs.
- The response time prediction results indicate the possibility of all the alternative architectures being potentially implemented using currently available microcontrollers. The ROM and RAM memory, and number of I/O estimations were also within the current microcontroller specifications.
- Due to the designer dependent program nature, although better results could statistically be gained, given more ECUs to predict and experiment on and hence get more generalised multiplying factor, the prediction confidence is not expected to be high. However, without any more system information, the author sees no alternative prediction method.
- Nonetheless, by measuring the actual response times of a large number of ECUs of each type, for example ABS ECU, a database of a particular system response times can be obtained. This information could be used to assist the prediction, since it provides the range of possible response times of each system as a prediction guideline.

- The microcontroller specification estimate was to be used for ECU component cost calculation in Chapter 6.

8.3 Simulation of CAN Message Transfer

CAN is an ISO standard network protocol for in-vehicle high speed data communication for passenger vehicles, and hence it is likely to be installed in most advanced future vehicles. For this reason, CAN associated delay in data transfer under different vehicle electronic architectures was simulated, to inspect CAN capability in delivering messages within timing constrain. The simulation of CAN message transfer yields the following conclusions:

- Under current level of message load, a CAN network is capable of providing in-vehicle data transfer for all of the alternative architectures with more than half of network capacity in reserve.

8.4 System Wiring Complexity and Component Costs

The study of architecture wiring complexity and component costs of alternative architectures was to evaluate the suitability of the architectures for implementation, from vehicle manufacturers' point of view. The evaluation concluded that:

- The results show that *Architectures 2-5* with network capability contain 15-25% fewer wires than *Architecture 1.* In term of wiring weights, *Architectures 3, 5* have a potential weight saving of approximately 20% over other architectures. The study also found that the wiring weight saving is almost consistent between small and large passenger vehicles.
- The system cost study shows that *Architecture 3* has potentially the cheapest component costs, with 19% saving on the most expensive *Architecture 5.* The other three architectures are expected to have comparable costs of approximately 6% higher than *Architecture 3.*
- The projected component costs are a guideline for current implementation of the alternative architectures. Other cost factors cannot be included due to the unavailability of data such as assembly methods and associated labour hours required, or cannot be quantified such as ECU development cost. However, their

effects and those of future trends are discussed in Chapter 6.

- The plot of ECU pins against peripheral ECU component counts, used to statistically predict the number of these components, is based on ECUs with through hole technology. The increasingly more popular surface mount technology would be likely to possess different pin/component ratios. Also as with general statistical studies, it is believed that the prediction accuracy could be improved, if more sampled ECUs were available.

8.5 Electronic Component Reliability

Component reliability prediction of the alternative electronic architectures was made to investigate their practicality and compare their potential reliability. The conclusions from this work are as follows:

- The two component reliability prediction techniques show similar results, suggesting that the proposed *Architecture 5* have potentially higher failure rates than the other architectures. This is primarily due to it having the largest number of ECUs.
- The predictions rely on the number of component prediction, which is based on the plot of the number of ECU pins against peripheral ECU component counts. As discussed before in Chapter 6, further samples of powertrain and dynamic control ECUs are needed to verify the prediction accuracy. From the collection of ECUs that the author inspected, the ECU pins and peripheral components show a clear evidence of correlation. However, with this small data set, the correlation can not be considered conclusive. If a large set of data is available and the relationship is firmly established, the plot also has potential applications in other areas such as ECU power estimation.
- Despite the wide use of handbooks for reliability predictions, there has been doubt about their accuracy in the aerospace industry [5]. There is a joint effort by manufacturers and regulators in the industry to develop a new reliability assessment program. This, however, has not yet been followed by the automotive industry. In spite of this doubt, the reliability prediction techniques used here should provide a useful early quantitative reliability assessment of any new system designs, especially in comparison to other designs.

8.6 Distributed Wheel Controller Architecture

Overall, the thesis proposed the Distributed Wheel Controller Architecture as a potentially attractive alternative vehicle electronic architecture. The architecture has been thought of for the suspension system in the literature [1], due to its potential in integrating each distributed suspension control ECU with its wheel actuator unit, and also in [2-4] which employ a distributed control ECU on each wheel of a brake-by-wire system. By integrating the functions of ABS, suspension and 4WS, all of which are wheel related control systems, in each Distributed Wheel ECU, the author anticipated its potential benefits in terms of vehicle dynamic, safety and ease of extra control functions addition.

From the results obtained in this thesis, the actual advantages and disadvantages of the Distributed Wheel Architecture (*Architecture 5*) were seen to be as follows:

- It has a longer worst case CAN message delay times for highest priority messages, than other architectures, due to its larger number of ECUs. However, its maximum delay time was still found to be well within the system requirement, and hence leaves CAN with spare capacity for message addition.
- Both *Architectures 3 and 5* contain noticeably fewer number of wires and less overall wiring length and weight than the rest. The results are consistent between small and large passenger vehicles.
- The proposed architecture component cost is more expensive than currently installed architectures. Its relatively high component cost at present may dampen its prospect on small vehicle market, which is highly cost sensitive.
- The proposed architecture has a potentially higher electronic component failure rate than the rest of the architectures, due to its larger number of ECUs.

Electronic control systems form a very important and safety related part of a vehicle. Any architectural change or new system introduction will have effects on many areas of the vehicles. Before a new architecture can be adopted, vehicle manufacturers will have to thoroughly research its worth in practicality, performance and financial aspects, on this exhaustive list of areas such as safety, assembly, model compatibility, development time and cost, reliability, regulations, customers' requirements, etc.

Hence, the proposed *Architecture 5* has functional performance benefits, but these are potentially overshadowed by cost and reliability concerned, as shown in this thesis. However, the cost situation is likely to change as electronics development continues, and the vehicle systems are also changing with introduction of new control systems and increased in functional integration among existing systems. These changes could make the *Architecture 5* more attractive in the future.

The effects of these changes should, therefore, be investigated further, as discussed in the next section.

8.7 Further Work

In addition to the research executed in this project, future work covering other electronic related aspects and functionality benefits of the proposed architecture in comparison to others, needs to be done in the following areas:

- With the benefit of networking, it is possible to add functions to the control systems using additional software without the need for extra ECUs. Furthermore, integrated control between systems for improved vehicle handling can be achieved, such as sharing signals between braking and steering functions to allow the driver to retain control while braking hard in a bend. Further work in this area is to study the algorithms of these possible operations such as combined ABS, active suspension and 4WS control, tyre pressure monitoring, and hill hold (automatically applying brakes to keep the car still, while it stops on a slope surface). Simulation could be done as in Chapter 3 to demonstrate these operations. Their software pseudo codes can be derived to estimate the resulting software overheads on the system ECUs, as carried out in the hardware performance prediction in Chapter 5. This would indicate how much current microcontroller capability is required for various potential functional additions.
- Investigate the functional benefits on vehicle safety of alternative architectures. Networking and combined vehicle dynamic control lead to closer collaboration among control systems. This could enable a safety upgrade from failsafe to fault tolerant system in some cases. For instance, a cruise control system with a broken vehicle speed sensor may receive the data from an ABS system and carry on its operation, which would otherwise be shut down. Another example is when one Wheel Controller ECU of *Architecture 5* is broken, instead of shutting all the

systems down, the other ECUs could potentially take over and continue some functions. ABS operation could be run albeit in degraded capacity, such as coupling the problematic wheel with the laterally opposite one, effectively running in 3-channel instead of a better 4-channel mode. An extent to which onboard diagnostics could be applied to each architecture is also of interest. This could be done by Failure Mode and Effect Analysis (FMEA) to analyse possible failure cases. This allows the study of possible action to each failure case, to enable fault tolerant system capability.

- Oncoming technology such as brake-by-wire and steer-by-wire are believed to yield great benefits in vehicle weight and component reduction. However, their success in implementation strongly depends on the safety of these heavily electronic based systems [6]. Redundancy systems may be introduced in *architectures 1-4*, but not necessary in *Architecture 5* since a *Wheel Controller* of one wheel could potentially resume the functions of another broken controller. These changes could, therefore, have significant effect on the alternative architectures in terms of vehicle assembly and cost. The implications on vehicle by having additional ECUs or multiple bus structures would incurred an extra cost, which should hence be evaluated. This would also provide the basis for comparison between different safety strategies.
- Investigate a means of predicting the usage of a specific microcontroller running a control function. This is a common problem and one for which, from literature studies, appears to be still unresolved. In practice, manufacturers use past experience and historical data on a similar microcontrollers running similar type of control functions as the basis for prediction. Without this information, the prediction method based on software pseudo code, developed in this project, gives mixed results. Further research could be to build a database of predictions for each type of control system to establish the accuracy range. In addition, the processor time ratio between functions such as signal manipulation, main control and diagnostics, could be explored by monitoring various ECU signals. This would enable a more detailed study of specific control sections for prediction.
- Recent development sees networking protocols, such as Local Interconnect Network, Time Triggered CAN and particularly Time Triggered Protocol (TTP), as alternatives to conventional CAN [2,7-9], especially for future by-wire systems.

TTP offers exact time schedule of each node for message transfer, and hence guarantees a deterministic message delay time [6] as well as eliminates bus overload problem [9]. Importantly, its fail-silence property can ensure that a faulty node detects itself and stops its message transmission, and hence avoids compromising vehicle safety with faulty data. Their impact on the alternative architectures should, therefore, be examined. Bus speed and message format should be studied. These characteristics, together with the list of modeled vehicle signals in Chapter 5, could be used to formulate the network implementation on the alternative architectures. The level of message load and potential room for addition can indicate network suitability for current and future installation on these architectures. Hardware required for the network implementation should also be investigated for its cost impact on the alternative architectures.

- Recently some vehicles contain several buses with different transfer speeds, dedicated for body electronics, chassis control and entertainment, which cross-communicate through gateways. The approach is introduced to ease bus congestion of a single bus vehicle, and hence improve on message delay. Simulation on message transfer would give approximate transfer delays and levels of message load for different architectures. These results, together with associated hardware cost, would provide the basis for comparison between CAN and the networks mentioned above, on the alternative architectures

8.8 References

1. *Ito H, et al.* **Controller for Experimental Vehicle Using Multi-Processor System** SAE No. 910086
2. *Hedenetz B, et al.* **Brake-By-Wire Without Mechanical Backup by Using a TTP-Communication Network** SAE No.981109
3. *Advanced Vehicle Systems Division, Motorola Semiconductor Products Sector* **'By Wire' Technology** Motorola, 2000
4. *Bannatyne R* **Electronic Braking Control Developments** Automotive Engineering International, February 1999
5. *Condra L, et al.* **Reliability Assessment of Aerospace Electronic Equipment** Quality and Reliability Engineering International Volume 15 Issue 4, 1999
6. *Dilger E, et al.* **Towards an Architecture for Safety Related Fault Tolerant Systems in Vehicles** European Conference on Safety and Reliability, 1997
7. *Quigley C, et al.* **An Investigation into the Future of Automotive In-Vehicle Control Networking Technology** SAE No.2001-01-0071
8. *Kopetz H, et al.* **TTP – A New Approach to Solving the Interoperability Problem of Independently Developed ECUs** SAE No.981107
9. *Heiner G, et al.* **Time-Triggered Architecture for Safety-Related Distributed Real-Time Systems in Transportation Systems** FTCS – Fault Tolerant Computer Symposium, 1998

APPENDIX A PREDICTION OF ECU CONTROL PSEUDO CODE AND ASSEMBLY PROGRAMS

Cruise Control

Throttle Demand Calculation Pseudo Code

Pseudo Code

```
i.    subtract vehicle speed to cruise speed to give e
ii.   table look up for gain $K_p$ based on the value of e
iii.  multiply $K_p$ to e
iv.   table look up for gain $K_I$ based on the value of ??
v.    multiply $K_I$ to $\sum_{m=1}^{M} E_{n-m}$
vi.   add the two products to get d
vii.  update $\sum_{m=1}^{M} E_{n-m}$
```

Throttle Demand Calculation Predicted Assembly Code

Program Code		*Execution Cycle*
---------	load cruise set speed	3
i	subtract vehicle and set speeds (get e(error value))	13
---------	save e in memory	3
	jump if carry not set (case e is +ve)	2
ii	jump if carry set (case e is -ve)	2
	complement A (convert e to +ve for table look up)	1
	A + 1 (complete conversion)	1
---------	table look up for K_p (when e is -ve)	10
	save K_p	3
iii	load e to A	3
	multiply K_p to e	588
---------	save K_p . e in memory	3
iv	load $\sum_{m=1}^{M} E_{n-m}$ from memory	3
---------	table look up for K_I	10
v	multiply $\sum_{m=1}^{M} E_{n-m}$ to K_I	588
---------	save value in register	3
	load carry counter address to register	2
	clear carry counter	1

	load lower 8 bits of K_p . e from memory	3
	load address of lower 8 bits of $K_I.\sum_{m=1}^{M} E_{n-m}$ to register	2
	add lower 8 bits with carry of K_p.e to $K_I.\sum_{m=1}^{M} E_{n-m}$	1
	save sum in memory	3
	jump if carry not set (no overflow)	2
	jump if carry set (overflow)	2
	increment carry counter	3
vi	load higher 8 bits of K_p . e from memory	3
	load carry counter address to register	2
	add carry counter to A	1
	load address of higher 8 bits of K_p.e to $K_I.\sum_{m=1}^{M} E_{n-m}$	2
	add higher 8 bits with carry of K_p.e to $K_I.\sum_{m=1}^{M} E_{n-m}$	1
	save sum in memory	3
	jump if d +ve	2
	output motor control signal	2
	output solenoid control signal	2
	jump if d -ve	2
	output motor control signal	2
	output solenoid control signal	2
---------	output to port	2
	load $\sum_{m=1}^{M} E_{n-m}$ from memory to register	3
	load e address to register	2
	add e to $\sum_{m=1}^{M} E_{n-m}$	1
vii	save sum in register	3
	subtract e_{n-m} from e + $\sum_{m=1}^{M} E_{n-m}$	13
	save new $\sum_{m=1}^{M} E_{n-m}$	3
	load e	3
---------	save e in memory (in place of e_{n-m})	3
	Total	**1307**

Note that 80C49 microcontroller has a limited set of instructions, compared to modern microcontrollers. For example, it does not have subtract instruction. The operation has to be carried out by combining a series of instructions. In the tasks and execution times above, such instructions were subtraction, table look up and multiply. The combination of instructions, which made up the tasks are given details below.

	Number of clock cycles
Load/Save/Add (between A and memory address)	
load memory address to register	2
load/save/add to/from/to A	1
	3

Table look up

Load table index to A	3
add first memory address of table to A	2
move a to register	1
move data from memory to A	1
save result	3
	10

subtraction (x-y)

clear carry	1
load y to A	3
complement A	1
A + 1 (= -y)	2
load x address to register	2
add x with carry to A (x-y)	1
save result	3
	13

Multiplication(8 bits x 8 bits)

The multiplication is programmed in the same way that the long multiplication is done by hand i.e. by multiplying each bit of one number to the other number. For example 4 bit x 4 bit multiplication of ABCD x 1011 is done by

$$\begin{array}{ccccccc}
 & & & A & B & C & D \\
 & & \times & 1 & 0 & 1 & 1 \\
\cline{4-7}
 & & & A & B & C & D \\
 & & A & B & C & D & \\
 & 0 & 0 & 0 & 0 & & \\
+A & B & C & D & & & \\
\hline
 & & & & & \multicolumn{2}{r}{\underline{\text{ANSWER}}}
\end{array}$$

The 8 bit x 8 bit multiplication program is constructed from a simpler 4 bits x 4 bits program.

The whole program code of 4 bits x 4 bits and 8 bits x 8 bits is shown below.

4 bit x 4 bit Multiplication (X x Y)

Program Code	***Execution Cycle***	
Data preparation		
load X	3	
save X to @0*	3	
load Y	3	
save Y to temp memory	3	
load X	3	$0000X_3X_2X_1X_0$
shift A left	1	$000X_3X_2X_1X_00$
save A to @1	3	
shift A left	1	$00X_3X_2X_1X_000$
save A to @2	3	
shift A left	1	$0X_3X_2X_1X_0000$
save A to @3	3	
clear A	1	
save A to @sum	3	
	<u>31</u>	
Multiplication		
{ load Y to A	3	
A AND 00000001	2	
jump if A zero	2	
jump if A not zero (bit 0 of Y is 1)	2	
load @sum	3	
add content of @0 to A	3	
save A to @sum	3	
	<u>18</u> }	
{ } is repeated 4 times (4 bits)	18 x 4 = 72	

Total cycles required for 4 bit x 4 bit multiplication is **<u>103</u>**

8 bit x 8 bit Multiplication

8 bit x 8 bit multiplication can be carried out by doing a number of 4 bit x 4 bit multiplication. This is done by splitting each 8 bit number into high 4 bits and low 4 bits and multiplying them. These result in 4 products, which can then be added up to get the result. An example of X x Y is as shown below.

$$\begin{array}{r} Y_7\,Y_6\,Y_5\,Y_4\quad Y_3\,Y_2\,Y_1\,Y_0 \\ \times\ \ \underline{X_7\,X_6\,X_5\,X_4\quad X_3\,X_2\,X_1\,X_0} \end{array}$$

Name the four most significant bits of Y, Y2 and the four least significant bits Y1. X2 and X1 are also named accordingly. The product of X and Y is:

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
									← Y1.X1 (=P1) →							
+					← Y2.X1 (=P2) →											
+					← Y1.X2 (=P3) →											
+	← Y2.X2 (=P4) →															
=	**X x Y**															

The program code is written below:

Program Code	***Execution Cycle***
load X	3
save X in temporary memory	3
load Y	3
save Y in temporary memory	3

Calculating Y1.X1 (=P1)

load X to A	3
A AND 00001111 (get X1)	2
save X1	3
repeat above to get Y1	8
Y1.X1 (P1)	103
Repeat the loop 3 times to get P2-4	119x3=357

<u>Addition</u>

clear A	1
load @sumlow address to register	2
save A to @sumlow (clear memory space for low 8 bit result)	1
repeat the above for @sumhigh	4
clear carry	1
clear counter	3

P1 + P2

load P2 to A	3
shift left 4 times (move P2 low 4 bit to add with high 4 bit of P1)	4
add P1 to A (P1+P2)	3
save result to @sumlow (lower 8 bit result)	3
jump if carry 0	2
jump if carry 1	2
increment counter in memory	3

clear carry	1
Add P3 to the sum	21
load P2 to A	3
shift right 4 times (using 4 high bits of P2 for high 8 bit addition)	4
save result	3
load P3 to A	3
shift right 4 times (using 4 high bits of P3 for high 8 bit addition)	4
add P2 high 4 bits to A	3
add carry counter to A	3
save result (P2+P3+carry)	3
clear carry counter	4
load P4 to A	3
add (P2+P3+carry) to A	3
save result to @sumhigh (higher 8 bit result)	3
jump if carry 0 (multiplication completed)	2
jump if carry 1 (overflow)	2
increment carry counter (keep record)	3
Total number of cycles taken by the 8 bits x 8 bits is	**588**

Predicted Assembly Programs of Other Processes

Program Code	*Execution Cycles*
Button press check	
load button reading from port to A	2
jump if 0 (no button pressed)	2
jump if not zero (button pressed)	2
	6
SET, RES, brake button pressed check	
load button reading from port to A	2
A AND value for SET	2
jump if SET	2
repeat the above 2 times for RES and brake checks	12
	18
Cruise status check	
load cruise status variable to A	3
A AND cruise on value	2
jump if zero (cruise off)	2

jump if not zero (cruise on)	2
	9

Vehicle speed check

load vehicle speed	3
subtract minimum speed to vehicle speed	13
jump if out of range	2
	18

Set reference(set) speed

load vehicle speed	3
store vehicle speed in set speed memory space	3
increment set speed set counter	3
	9

Check if reference(set) speed is previously set

load set speed set counter	3
jump if not zero (set speed has been set)	2
jump if zero (set speed has not been set)	2
	7

Response Time Prediction

Response time is predicted by adding the cycle times, of all the processes the cruise control ECU executes in response to SET, as shown in Cruise flowchart in Figure 3 of Chapter 5. The total cycle counts in response to SET is *1413.*

The 80C49 microcontroller has a machine cycle = (1 ÷ crystal clock frequency) x 15

The circuit clock is 8 MHz

Hence the machine cycle is *1.875* µs

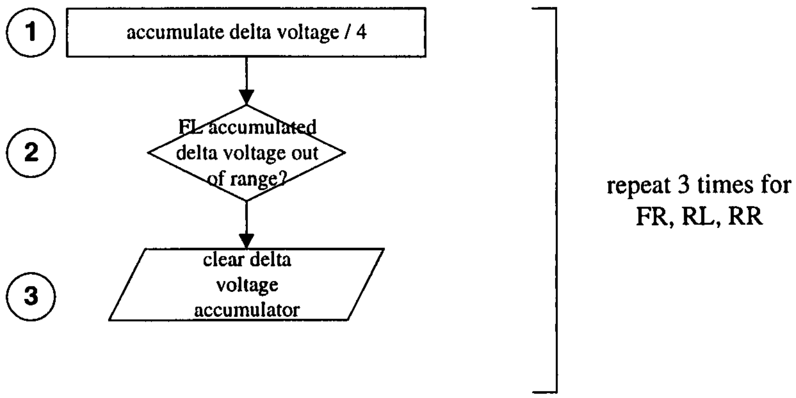
∴ The predicted response time to SET is

1413 x 1.875µs **= 2.65 ms**

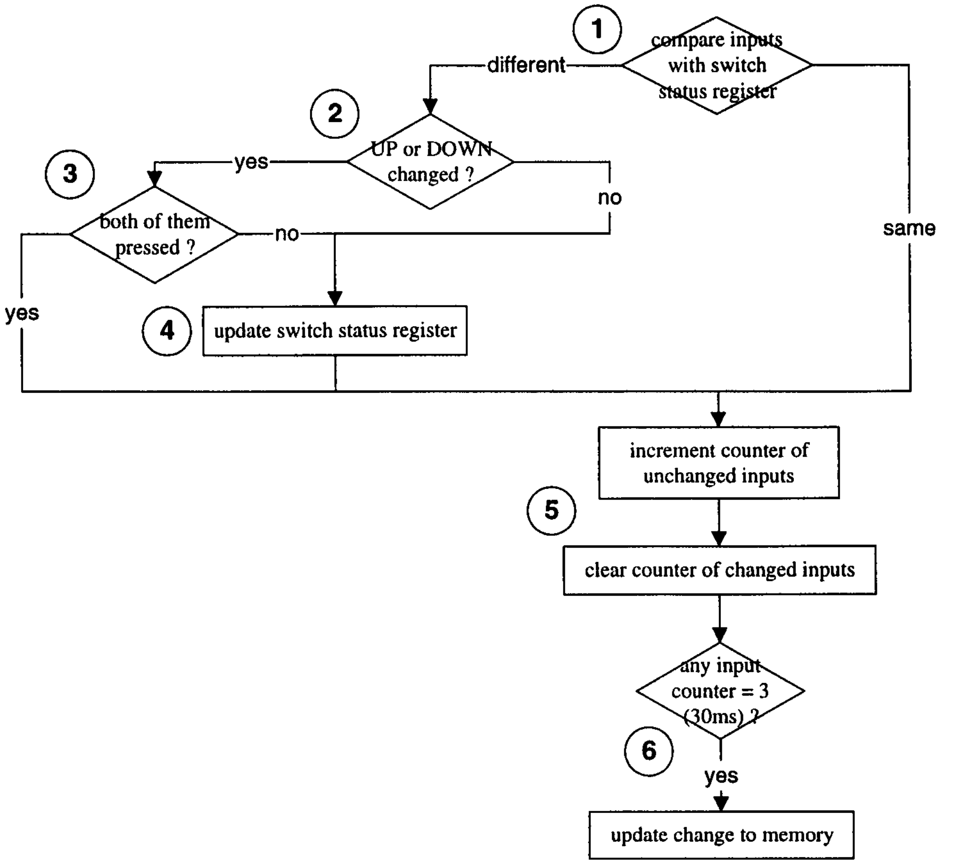
Air Suspension

Air Suspension Flow Charts

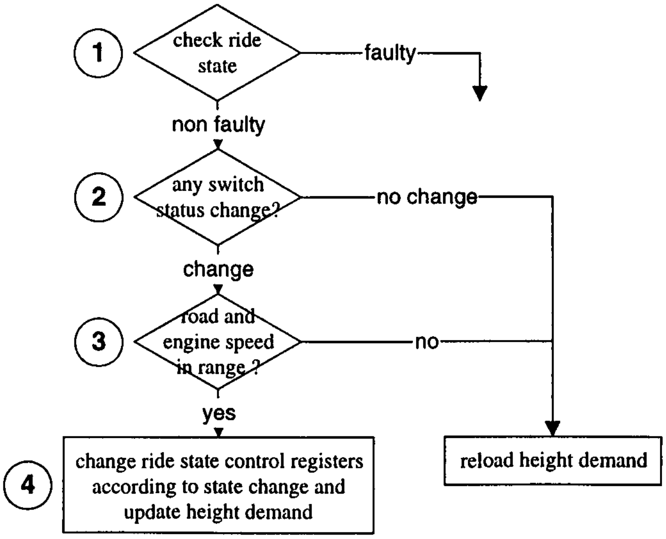
1 Calculate Air Spring Heights


1
accumulate delta voltage / 4
2
FL accumulated delta voltage out of range?
3
clear delta voltage accumulator
repeat 3 times for FR, RL, RR

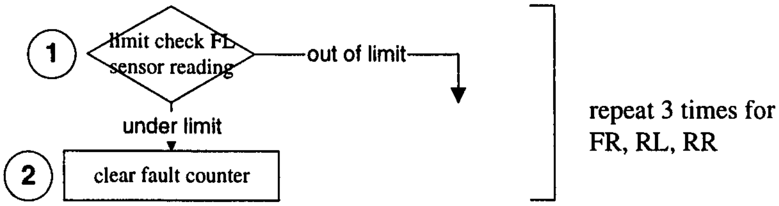
2 Debounce Vehicle Status Logic


1
compare inputs with switch status register
different
same
2
UP or DOWN changed ?
yes
no
3
both of them pressed ?
no
yes
4
update switch status register
increment counter of unchanged inputs
5
clear counter of changed inputs
any input counter = 3 (30ms) ?
6
yes
update change to memory

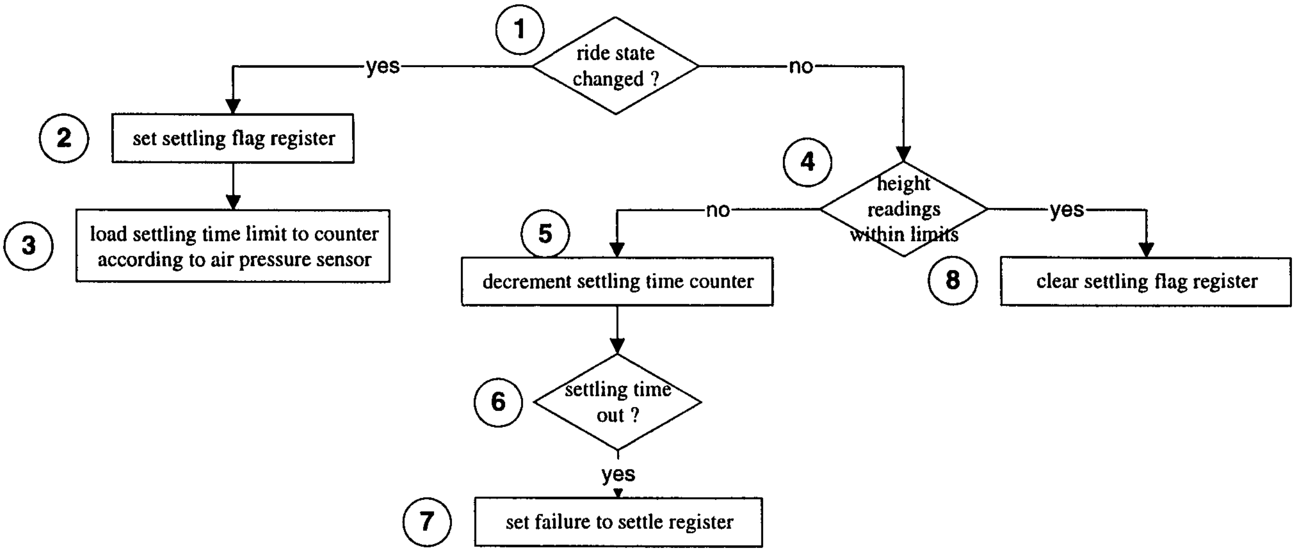
3 Process Vehicle Status Logic


1
check ride state
faulty
non faulty
2
any switch status change?
no change
change
3
road and engine speed in range ?
no
yes
4
change ride state control registers according to state change and update height demand
reload height demand

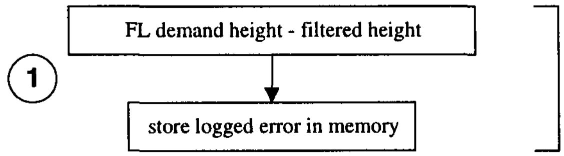
4 Check for Sensor Faults



5 Settle at Datum Analysis

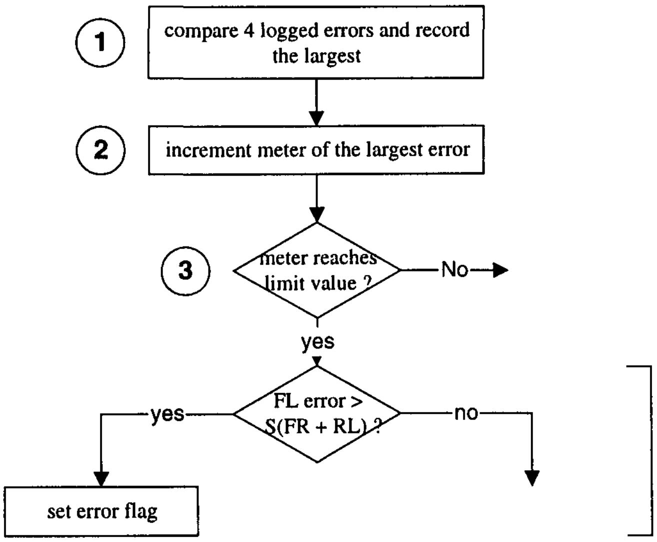

1
ride state changed ?
yes
no
2
set settling flag register
3
load settling time limit to counter according to air pressure sensor
4
height readings within limits
no
yes
5
decrement settling time counter
8
clear settling flag register
6
settling time out ?
yes
7
set failure to settle register

6 Calculate Air Spring Logged Errors


FL demand height - filtered height
1
store logged error in memory

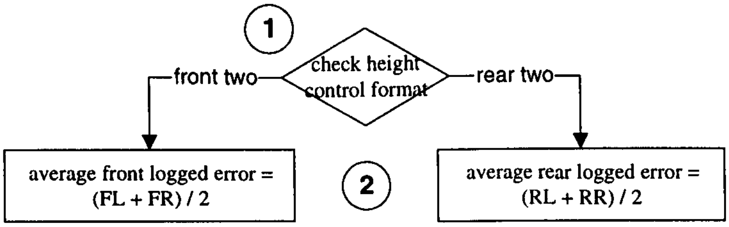
repeat 3 times for
FR, RL, RR

7 Increment Excess Error Meters

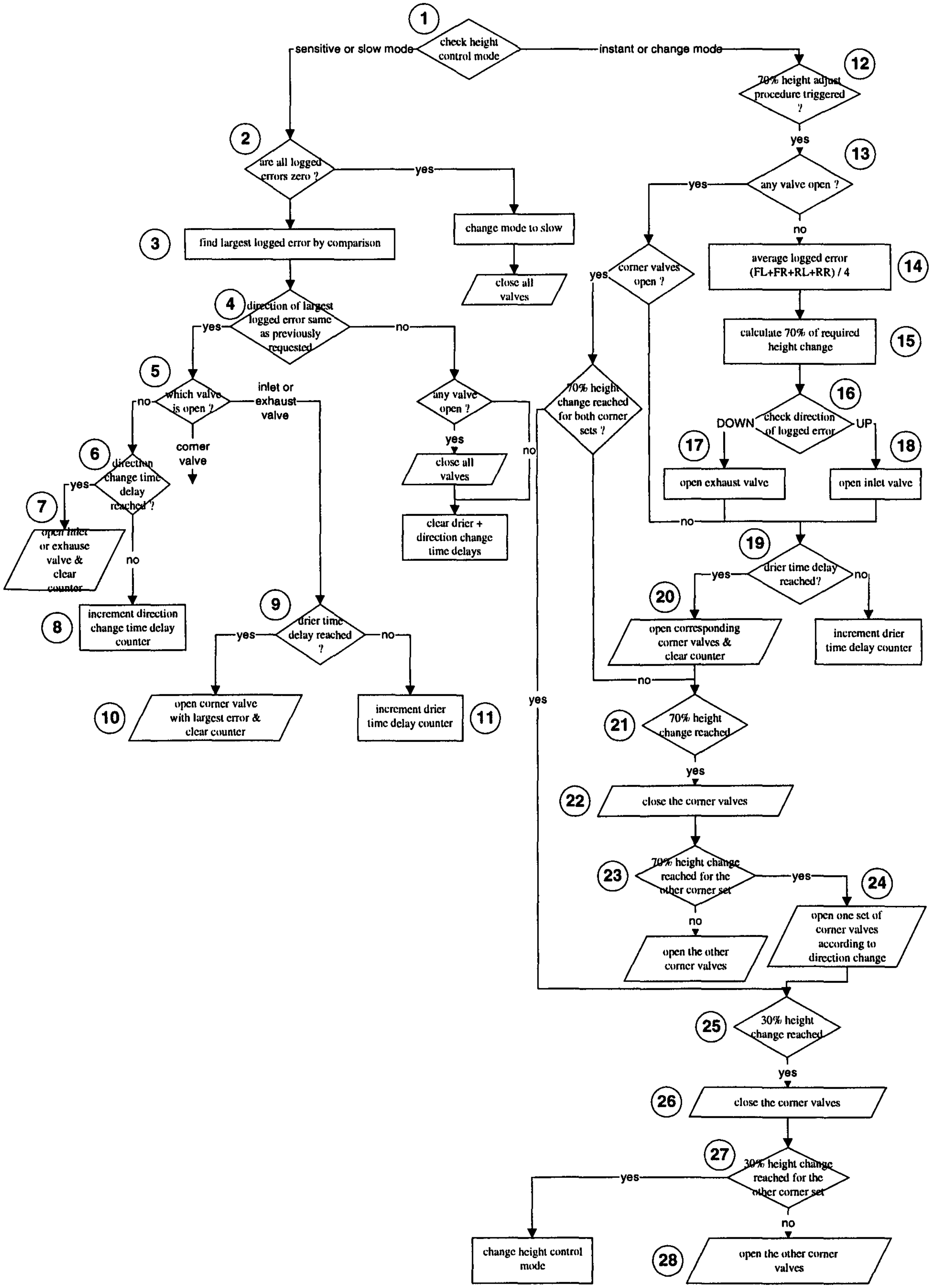

1
compare 4 logged errors and record the largest
2
increment meter of the largest error
3
meter reaches limit value ?
No
yes
yes
FL error > S(FR + RL) ?
no
set error flag

repeat 3 times for
FR, RL, RR
S - sensitivity constant

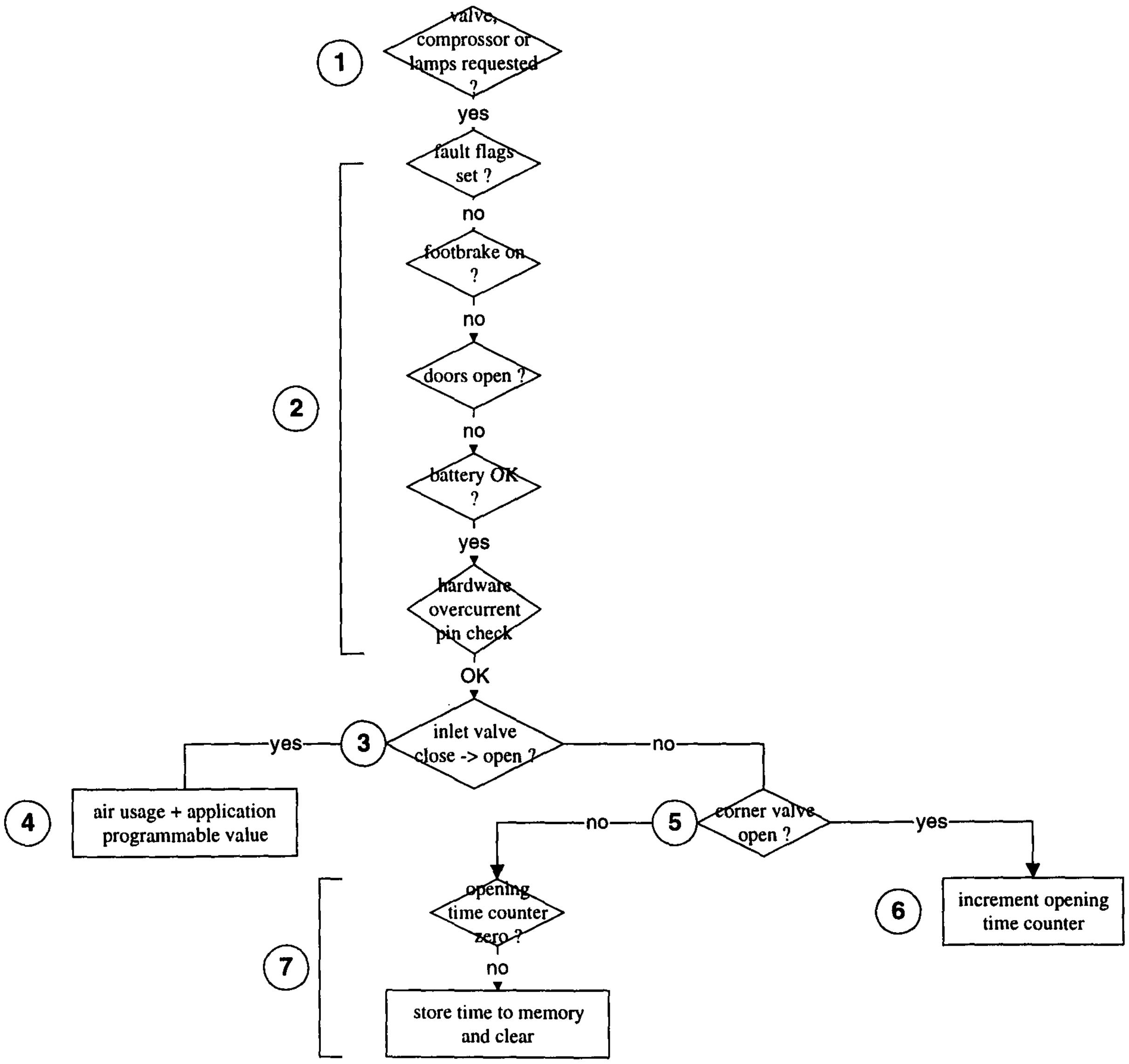
8 Average Logged Errors


1
front two
check height control format
rear two
average front logged error = (FL + FR) / 2
2
average rear logged error = (RL + RR) / 2

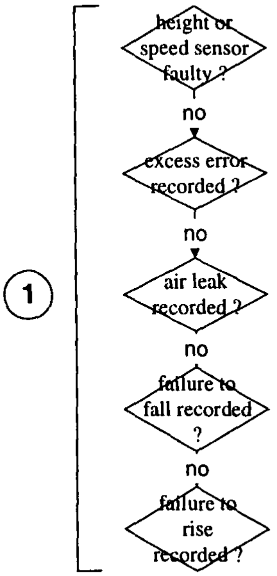
9 Adjust Air Spring Heights


1
check height control mode
sensitive or slow mode
instant or change mode
2
are all logged errors zero ?
yes
change mode to slow
close all valves
3
find largest logged error by comparison
4
direction of largest logged error same as previously requested
yes
no
5
which valve is open ?
no
inlet or exhaust valve
corner valve
any valve open ?
yes
no
close all valves
clear drier + direction change time delays
6
direction change time delay reached ?
yes
no
7
open inlet or exhause valve & clear counter
8
increment direction change time delay counter
9
drier time delay reached ?
yes
no
10
open corner valve with largest error & clear counter
11
increment drier time delay counter
12
70% height adjust procedure triggered ?
yes
13
any valve open ?
yes
no
corner valves open ?
yes
no
70% height change reached for both corner sets ?
yes
no
14
average logged error (FL+FR+RL+RR) / 4
15
calculate 70% of required height change
16
check direction of logged error
DOWN
UP
17
open exhaust valve
18
open inlet valve
no
19
drier time delay reached?
yes
no
20
open corresponding corner valves & clear counter
increment drier time delay counter
21
70% height change reached
yes
22
close the corner valves
23
70% height change reached for the other corner set
yes
no
24
open one set of corner valves according to direction change
open the other corner valves
25
30% height change reached
yes
26
close the corner valves
27
30% height change reached for the other corner set
yes
no
change height control mode
28
open the other corner valves

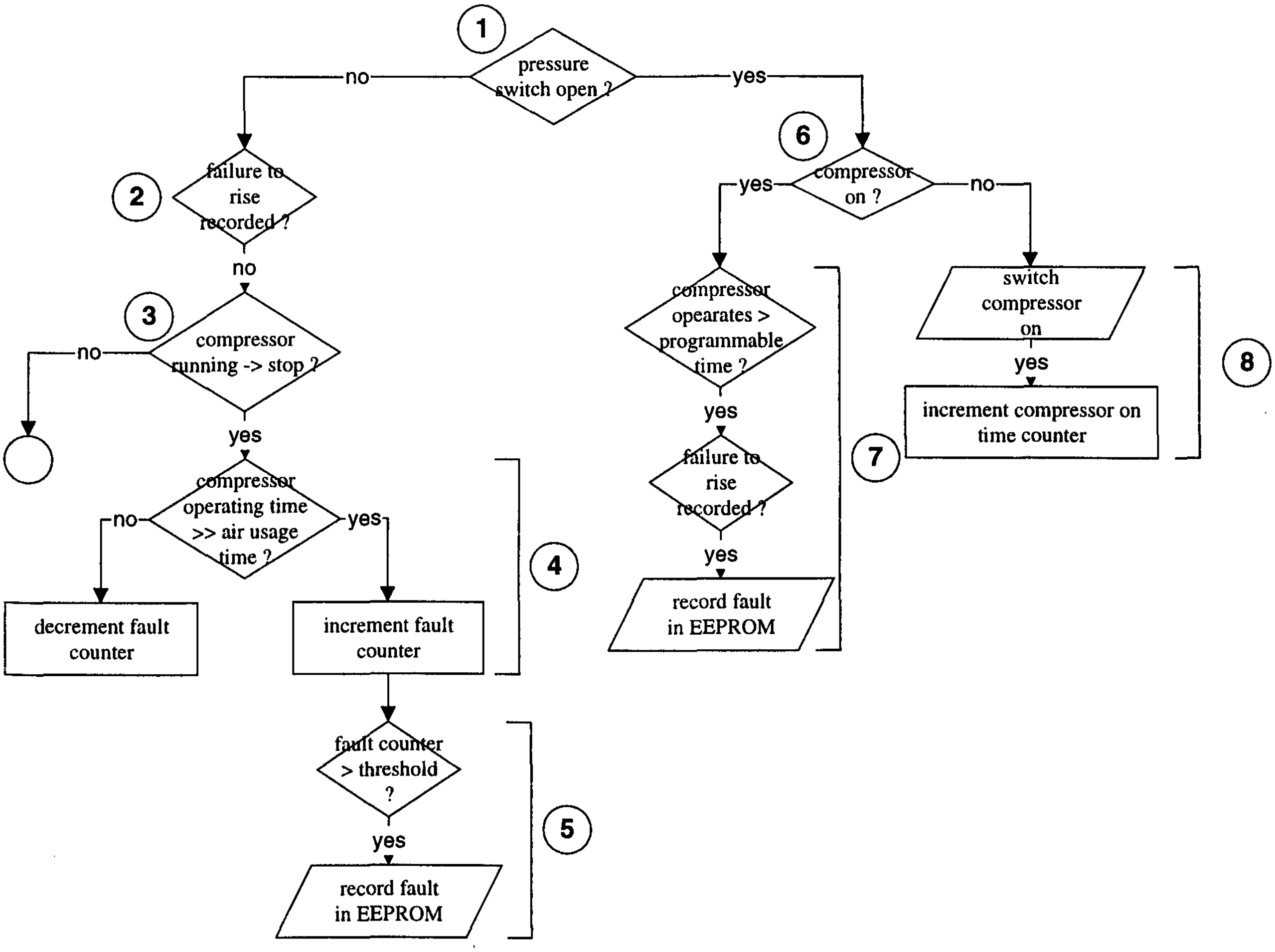
10 Drive Outputs and Select Modes


1
valve, compressor or lamps requested ?
yes
2
fault flags set ?
no
footbrake on ?
no
doors open ?
no
battery OK ?
yes
hardware overcurrent pin check
OK
3
inlet valve close -> open ?
yes
4
air usage + application programmable value
no
5
corner valve open ?
no
yes
7
opening time counter zero ?
no
store time to memory and clear
6
increment opening time counter

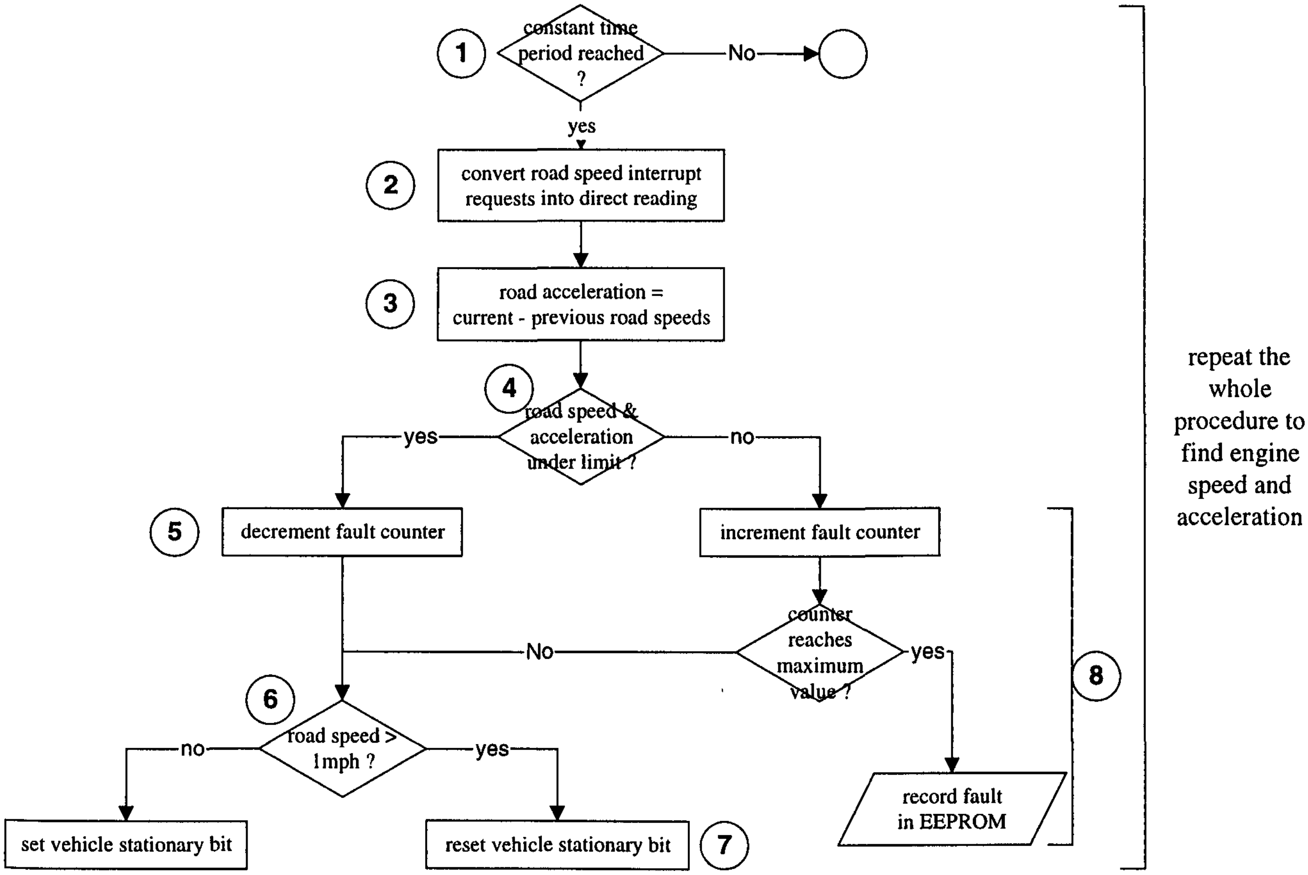
11 Suspension Fault Modes


1
height or speed sensor faulty ?
no
excess error recorded ?
no
air leak recorded ?
no
failure to fall recorded ?
no
failure to rise recorded ?

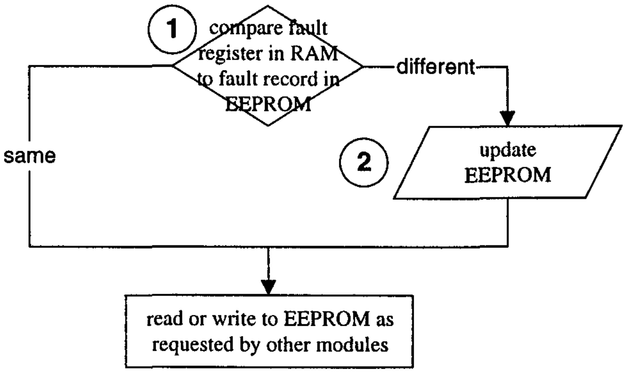
12 Regulate Compressor Air Pressure


1
pressure switch open ?
no
yes
2
failure to rise recorded ?
no
3
compressor running -> stop ?
no
yes
compressor operating time >> air usage time ?
no
yes
decrement fault counter
increment fault counter
4
fault counter > threshold ?
yes
record fault in EEPROM
5
6
compressor on ?
yes
no
compressor opearates > programmable time ?
yes
failure to rise recorded ?
yes
record fault in EEPROM
7
switch compressor on
yes
increment compressor on time counter
8

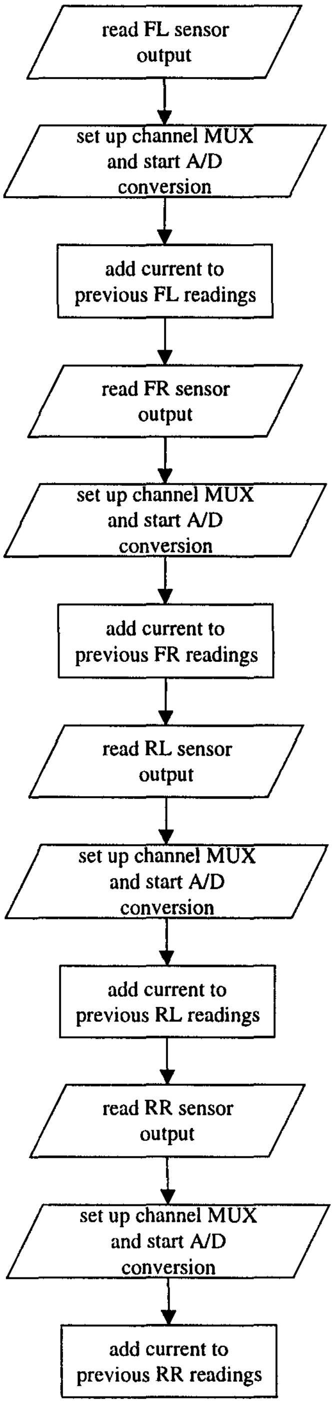
13 Calculate and Process Road and Engine Speed


1
constant time period reached ?
No
yes
2
convert road speed interrupt requests into direct reading
3
road acceleration = current - previous road speeds
4
road speed & acceleration under limit ?
yes
no
5
decrement fault counter
increment fault counter
counter reaches maximum value ?
No
yes
6
road speed > 1mph ?
no
yes
set vehicle stationary bit
reset vehicle stationary bit
7
record fault in EEPROM
8
repeat the whole procedure to find engine speed and acceleration

14 Program EEPROM


1
compare fault register in RAM to fault record in EEPROM
different
same
2
update EEPROM
read or write to EEPROM as requested by other modules

Measure Sensor Outputs (Synchronous Job)


read FL sensor output
set up channel MUX and start A/D conversion
add current to previous FL readings
read FR sensor output
set up channel MUX and start A/D conversion
add current to previous FR readings
read RL sensor output
set up channel MUX and start A/D conversion
add current to previous RL readings
read RR sensor output
set up channel MUX and start A/D conversion
add current to previous RR readings

Air Suspension Predicted Assembly Code

Air Suspension Sequential Job Modules

❶ *Calculate Air Spring Height*

		Cycles	
1	FL accumulated delta voltage ÷ 4	19	
	store FL	5	***24***
2	{compare FL to lower limit	4	
	branch if outside limit	3}	
	same to higher limit	7	$\boxed{14}$
3	clear high byte of accumulated voltage	5	
	〃 low 〃 〃 〃	5	$\boxed{10}$
same for FR, RL, RR		48x3=***144***	
Total		**192**	

÷ 4	load high byte to accumulated delta voltage	4
	rotate right through carry	3
	load low byte	4
	rotate right through carry	3
	clear carry	2
	rotate right through carry	19

❷ Debounce Vehicle Status Logic

		Cycles	
1	load switch inputs	4	
	compare to switch status register	4	
	branch if different	3	*9*
2	load switch status register to X	4	
	clear other switches in A (A AND UP&DOWN)	2	
	″ ″ X (X AND UP&DOWN)	2	
	store X in M	5	
	check if different (A ⊕ M)	4	
	branch if different	3	*17*
3	load switch inputs	4	
	clear other switches in A except UP (A AND UP)	2	
	branch if 0	3	
	clear other switches in A except DOWN (A AND DOWN)	2	
	branch if 0	3	*14*
4	store switch input	5	*5*
5	load switch change register	4	
	{compare to UP change (A AND UP)	2	
	branch if unchanged	3	
	increment unchanged counter	5}	
	clear unchanged counter (if changed)	(5)	
	same to DOWN, INHIBIT, FOOT, HAND-BRAKES, DOORS, AIR PRESSURE	6x10=60	*74*
6	{load UP counter	4	
	compare to 3	2	
	branch if equal	3}	
	set/reset UP bit in memory	5	
	same to DOWN, INHIBIT, FOOT, HAND-BRAKES, DOORS, AIR PRESSURE	6x14=784	*98*
Total		**222**	

❸ *Process Vehicle Status Logic*

		Cycles	
1	{load current ride state	4	
	compare fault ride state with current ride state	4	
	branch if current state is fault state	3}	
	check for other 2 fault states	2x11=22	***33***
2	{load request change register	4	
	compare to UP demand	4	
	branch if UP demand is requested	3}	
	same to DOWN	11	***22***
3	check if road speed not faulty (same as ❸1)	11	
	″ engine speed ″ ″	11	***22***
4	check if ride state is kneel (same as ❸1)	11	
	″ ″ standard, low, high, belly out	4x11=44	
	{load new ride state	4	
	store new ride state	5}	
	load and store height adjust procedure, settle at datum analysis, height control mode, direction time, drier time, and FL, FR, RL, RR demands	9x9=81	***145***
Total		**222**	

❹ *Check for Sensor Faults*

1	{compare FL reading to higher and lower limit (same as ❸1)	2x11=22	
2	clear fault counter	5}	
Repeat 1, 2 for FR, RL, RR		3x27=81	***108***
Total		**108**	

❺ Settle at Datum Analysis

		Cycles	
1	check if ride state has changed (same as ❸1)	11	11
2	set settling flag register	5	5
3	check if air pressure very low	11	
	" " " just below cut off line	11	
	" " " above cut off line	11	
	load settling time limit according to air pressure	4	
	store in settling time limit counter	4	45
4	{check if height reading within upper limit	11	
	" " " lower limit	11	22
5	decrement settling time counter	5	5
6	check if settling time counter is 0	11	11
7	set failure to settle register	5}	5
	same for the other 3 corners	3x43=129	129
8	clear settling flag register	5	5
	Total = process 1 + 4 + 5 + 6 + 7 (the longest route)	**183**	

❻ Calculate Air Spring Logged Errors

1	{load demand height of FL	4
	demand - filtered heights (=logged error)	4
	branch if negative	3
	set sign bit	5
	complement logged error	3
	clear carry	2
	store logged error	5}
	same to FR, RL, RR	78
	Total	**104**

❼ Increment Excess Error Meters

		Cycles	
1	load FR to A	4	
	X = 1 (load 1 to register X)	2	
	FR – FL	4	
	branch if equal	3	
	branch if lower	3	
	load FL to A	4	
	X = 2	2	
	FL – RR	4	
	branch if equal	3	
	branch if lower	3	
	load RR to A	4	
	X = 3	2	
	RR – RL	4	
	branch if equal	3	
	branch if lower	3	
	X = 4	2	***50***
2	{X → A	2	
	A – 4	2	
	branch if equal to 0	3}	
	repeat with A – 3, A – 2, A – 1	3x7=21	
	increment FR record in memory (FR has the greatest error)	5	***33***
3	compare largest error meter to limit	11	***11***
	(load 3		
	meter-limit 4		
	branch 3)		
4	{load FR error	4	
	load S	4	
	FR error + RL error	4	
	(FR error + RL error) x S	11	
	compare to FL	4	
	branch if lower	3	
	set error flag	5}	***35***
	same for FR, RL and RR errors	3x35=105	***105***
Total		**234**	

❽ Average Logged Errors

1	load ride mode	4	
	compare with front two mode	4	
	branch if in front two mode	3	
	" " rear "	3	***14***
2	load FL sign bit	4	
	compare to FR sign	4	
	branch if equal (same sign)	3	
	compare FR sign to positive	2	
	branch if positive	3	
	load FR logged error (-ve)	5	
	FR - FL	4	
	branch if negative	3	
	set 1 to sign bit	5	
	complement	3	
	(FL + FR) ÷ 2 (rotate right)	3	
	store average logged error	5	***44***
Total		**58**	

❾ Adjust Air Spring Heights

1	load height control mode	4	
	check if sensitive mode (compare & branch)	7	
	check if slow mode	7	***18***
2	{load FL logged error	4	
	compare to zero	4	
	branch if equal	3}	
	same for FR, RL, RR	3x11=33	***44***
3	load sign of largest logged error	4	***4***
4	compare the 2 directions (A ⊕ previous sign of largest error)	4	
	branch if different	3	
	load current largest logged error sign	4	
	store " " " "	5	***16***
5	load valve register	4	
	compare to zero	4	
	branch (to 6) if no valve open (0)	3	
	A AND corner valve open demands	4	
	branch if not zero (corner valve open)	3	***18***
6	load direction change time delay counter	4	
	compare to limit	4+3	***11***
7	check if largest logged error negative (load + branch)	4+3	***11***
	open inlet or exhaust valve	5	
	clear delay counter	5	***17***
8	increment counter	5	***5***
9	check if drier time delay reached	11	
	load valve open demand	4	***15***
10	A OR largest error indication data	4	
	store in valve open demand	5	
	clear drier time delay counter	5	***14***
11	increment counter	5	***5***
12	load ride mode	4	
	branch if it is 70% height adjust procedure	3	***7***
13	load valve open demand	4	
	branch if valve open	3	***7***
14	clear A	3	

{ check if FL error negative	4+3	
A - FL (FL negative), A + FL (FL positive)	4 }	
same for FR, average (RL&RR)	2x11=22	
branch if negative	3	
set sign bit	5	
complement A	3	
A ÷ 2 (rotate A right 3, clear carry 2)	5	
A ÷ 2	5	
store A	5	***62***
15 load 7 to X	2	
A x 7	11	
store A x 7	5	
(A x 7) ÷ 10	198	***216***
16 check logged error direction	4+3	***7***
17/18 open inlet/exhaust valve (set bit)	5	***5***
19 check if drier time delay reached	11	***11***
20 same as 10	14	***14***
21/23/25/27 check if height change reached	4+4+3=11	***44***
22/26/28 close both corner valves	5+5=10	***30***
24 open corner valves	10	***10***
Total = process 1 + 12 + ... + 28 - 18		**431**

❶⓪ ***Drive Outputs and Select Modes***

1	check if valve open requested	7	
	same for lamp, compressor	2x7=14	***18***
2	check if fault flag set	7	
	same for footbrake, doors, battery, hardware overcurrent pin	4x7=28	$\boxed{35}$
3	load valve open demand	4	
	A AND inlet valve open	2	
	branch if not open	3	
	compare to previous inlet valve state	4	
	branch if close → open	3	***16***
4	load air usage	4	
	A + programmable value	4	
	store air usage	5	***13***
5	load valve open demand	4	
	A AND corner valve open demand	2	
	branch if not open	3	***9***
6	increment counter	5	***5***
7	check if opening time counter zero	11	
	store time	5	
	clear counter	5	***21***
Total	= process 1 + 2 + 3 + 5 + 7	**102**	

❶❶ ***Suspension Fault Modes***

1	check if height or speed sensors faulty	7	
	check if other errors occur	4x7=28	$\boxed{35}$
Total		***35***	

❶❷ *Regulate Compressor Air Pressure*

1	check if pressure switch open	7	***7***
2	load flag register	4	
	A AND failure to rise recorded	2	
	branch if fail	3	$\boxed{9}$
3	check if compressor change from run → stop (= ❶⓿ 3)	16	***16***
4	load compressor operating time	4	
	operating time - air usage time	4	
	compare result to threshold	4	
	branch if smaller	3	
	increment counter	5	$\boxed{20}$
5	compare value in fault counter to limit	4+4+3=11	$\boxed{11}$
	(record result in EEPROM)		
6	check if compressor on (= ❶❷ 2)	9	***9***
7	check if compressor operates > programmable time	4+4+3=11	
	check if failure to rise recorded (=❶❷ 2)	9	$\boxed{20}$
	(record result in EEPROM)		
8	switch compressor on	5	
	increment compressor on time counter	5	***10***
Total = process 1 + … + 5		**63**	

❶❸ *Calculate and Process Road and Engine Speeds*

1	check if constant time period reached (= ❾6)	11	***11***
2	load road speed interrupt counter	4	
	store as speed	5	***9***
3	acceleration = A - previous road speed	4	
	store acceleration	5	***9***
4	check if road speed under limit	11	
	" acceleration "	11	$\boxed{22}$
5	decrement counter	5	$\boxed{5}$
6	load road speed	4	
	speed - 1	2	
	branch if speed < 1 mph	3	***9***

7 set/reset vehicle stationary bit		5	***5***
8 check if counter reaches maximum limit (=❶❸ 4)		11	$\boxed{11}$
Total		**81**	

❶❹ ***Program EEPROM***

1 { load fault EEPROM fault register status		4	
compare to that in RAM		7}	
same for sensor fault register		11	$\boxed{22}$
2 update EEPROM		24	$\boxed{24}$
(set bit in EEPROM control register	5		
load data to write to EEPROM from RAM	4		
write to EEPROM	5		
set bit in EEPROM control register	5		
reset the bit after program time elapsed	5)		
same for sensor fault register			
Total		**46**	

Air Suspension Synchronous Job Modules

A/D conversion		***27***
{ load A/D channel select and enable value	4	
store value to A/D control register (initiate A/D conversion)	5	
delay loop (wait for A/D conversion to complete)		
load A/D control register	4	
AND with A/D completion constant (check if conversion finished)	2	
branch if A/D has completed	3	
load A/D conversion result	4	
store result	5	
}		
same to FR, RL, RR	3x27=81	***81***
[repeat 3 more times (each sensor is read 4 times)	3x108=324	***324***

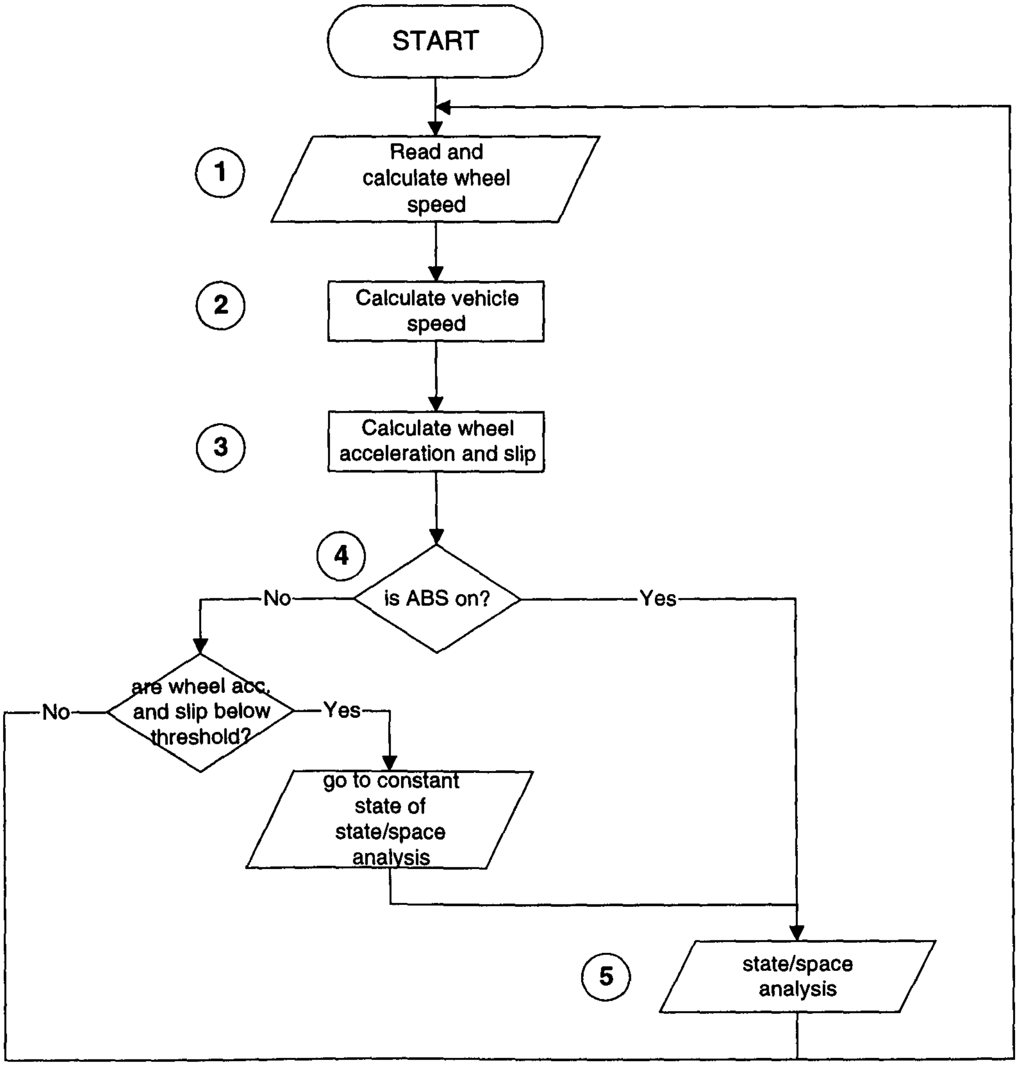
Total diagnostic codes	**895**	**cycles**
Total other codes	**1618**	**cycles**
Total codes	**2513**	**cycles**
Percent of diagnostic codes to total codes	**35.6%**	
Percent of diagnostic codes to other codes	**55.3%**	

Notes

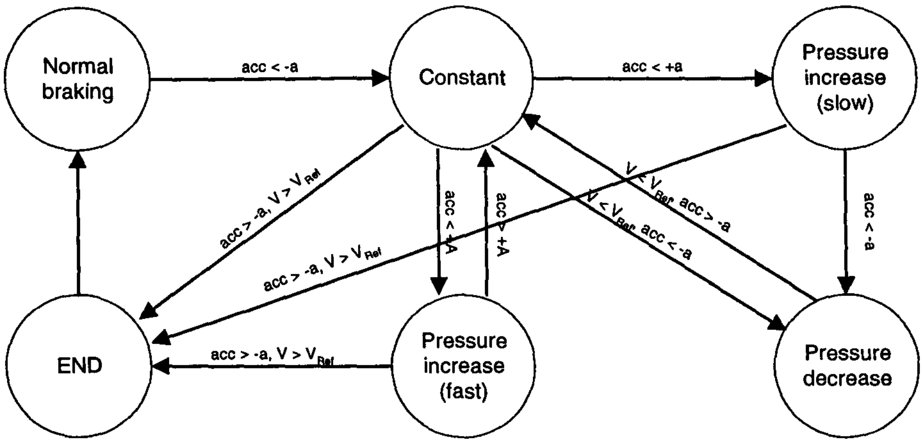
- □ represents diagnostic codes

ABS

ABS Flow Chart


START
1
Read and calculate wheel speed
2
Calculate vehicle speed
3
Calculate wheel acceleration and slip
4
is ABS on?
No
Yes
are wheel acc. and slip below threshold?
No
Yes
go to constant state of state/space analysis
5
state/space analysis

ABS State/Space Analysis



ABS Pseudo Code

❶ Read and calculate wheel speed

```
read wheel speed sensor inputs (A/D conversion)*
[
 {
        load A/D channel select and enable values
        store values to A/D control register (initiate A/D conversion)
        delay loop (wait for A/D conversion to complete)
        load A/D control register
        AND with A/D completion constant (check if conversion finished)
        branch if A/D has completed
        load A/D conversion result
        store result
 }

        divide wheel speed signal by 10 (scale down)**
]

[] x 4 (4 wheel speed sensors)
```

❷ Calculate vehicle speed

```
load FL wheel speed
FL + RR wheel speed
store result
load FR
FR + RL
{
        compare (FR + RL) to (FL +  RR)
        branch if (FR + RL) smaller
}***
load higher pair to A
shift right (÷ 2)
store result
```

❸ Calculate wheel acceleration and slip

```
{
        load current wheel speed
        current – previous wheel speed (= wheel acceleration****)
        store result
        load vehicle speed
        vehicle speed – wheel speed
        load 100 to another accumulator
        (vehicle speed – wheel speed) x 100
        divide the result with vehicle speed (= wheel slip percentage)
        store result
}

{ } x 4 (for 4 wheels)
```

❹ ABS status check

```
{
        load ABS status flag
        compare to ABS ON value
        branch if ABS is on
}*****

{ } x 4 (for 4 wheels)
```

❺ State/space analysis

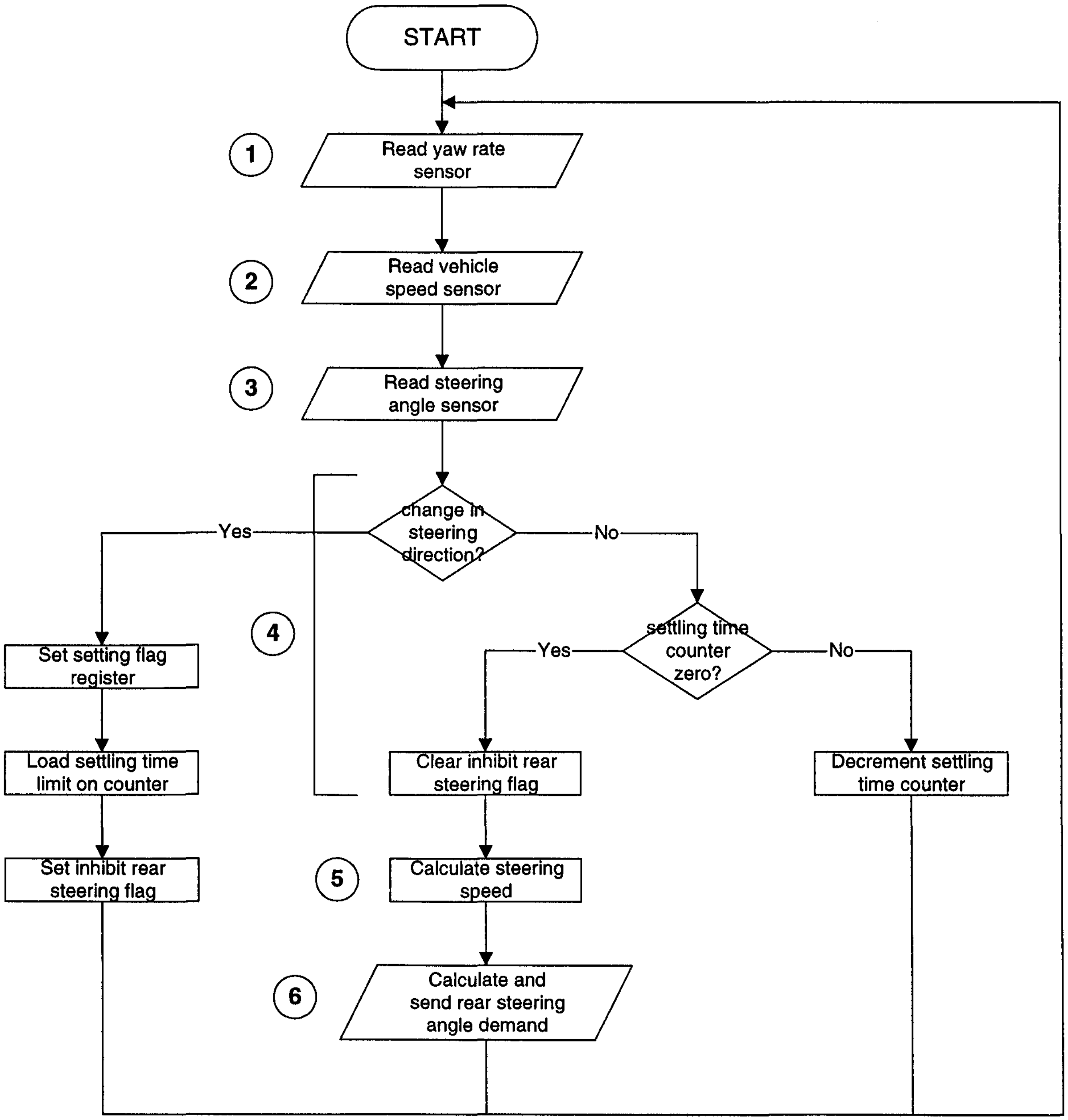
```
{       find out the current ABS state
load ABS state variable
compare ABS state variable to 'normal braking state' value
        x 4 (also compare it to the other 4 possible state values)

        find out the new ABS state
check if acc (wheel acceleration) exceeds a threshold value for state change
check if wheel speed exceeds a threshold value for state change
load appropriate state value
store the value as current ABS state variable
set brake valve register (according to the brake control law of the ABS state)}

        { } x 4 (for 4 wheels)
```

Asterisk(s) Explanation

* This { } is a typical '*read*' (analogue sensor input) program codes
** According to the ABS wheel speed sensor range given in [1]
*** This { } loop is a typical '*compare*' program codes
**** The time interval between successive readings of each wheel speed sensor is assumed fixed. Hence there is no need to divide successive wheel speed subtraction with this time, to obtain a wheel acceleration.
***** This { } loop is a typical '*check*' program codes

4WS

4WS Flow Chart


START
1
Read yaw rate sensor
2
Read vehicle speed sensor
3
Read steering angle sensor
change in steering direction?
Yes
No
4
settling time counter zero?
Yes
No
Set setting flag register
Load settling time limit on counter
Set inhibit rear steering flag
Clear inhibit rear steering flag
Decrement settling time counter
5
Calculate steering speed
6
Calculate and send rear steering angle demand

4WS Pseudo Code

❶ Read yaw rate sensor

```
read yaw rate sensor (same as ABS ❶ but without division)
```

❷ Read vehicle speed sensor

```
read vehicle speed sensor (same as ABS ❶ but without division)
```

❸ Read steering angle sensor

```
read steering angle sensor (same as ABS ❶ but without division)
```

❹

```
check if steering direction has changed
check if settling time counter zero (check if steering direction is definitely
        changing)******
clear inhibit rear steering flag
```

❺ Calculate steering speed

```
load current steering angle
current -  previous steering angle
store steering speed (see ****)
```

❻ Calculate and send rear steering angle demand

rear steering angle control calculation according to the control equation below (described in exemplary 4WS system in Chapter 3)

$$\theta_r = K_1(V).\theta_f + K_2(\dot{\theta}_f, V).K_3(V).\omega_y$$

$\dot{\theta}_f$: front wheel steering angle velocity
θr : rear wheel steering angle
θf : front wheel steering angle
V : vehicle speed
ω_y : yaw rate
$K_1(V)$: opposite direction steering angle proportional gain
$K_2(\dot{\theta}_f, V)$: tuning gain of steering velocity
$K_3(V)$: yaw rate proportional gain

```
[
        Table look up for K1(V)
 {
        load vehicle speed to index register
        load table content (index addressing)
}*******
        load front wheel steering angle (θf)
        multiply front wheel steering angle with gain ($K_1(V).\ \theta f$)
        store result
]

do $\omega_y.\ K_3(V)$ (same as [] above)
```

Table look up for $K_2(\dot{\theta}_f, V)$

```
load current vehicle speed (V)
load vehicle speed interval value of the table
current vehicle speed ÷ vehicle speed interval (to find row that V belongs)
store result
load current steering speed ($\dot{\theta}_f$)
load steering speed interval value of the table
current steering speed ÷ steering speed interval (to find column that $\dot{\theta}_f$ belongs)
store result
load total number of columns
load $\dot{\theta}_f$ column position
total number of columns x $\dot{\theta}_f$ column position
result + starting memory location for table (locate beginning of $\dot{\theta}_f$ column position in
        ROM)
result + V row position (locate the right gain in the table in index register)
Load table content (index addressing)
```

Calculation

```
Load $K_3(V).\ \omega_y$
$K_3(V).\ \omega_y$ x $K_2(\dot{\theta}_f, V)$
Load $K_1(V).\theta_f$
($K_1(V).\theta f$) x $K_3(V).\ \omega_y$ x $K_2(\dot{\theta}_f, V)$ ( = $\theta_r$ rear steering angle)
save $\theta_r$
```

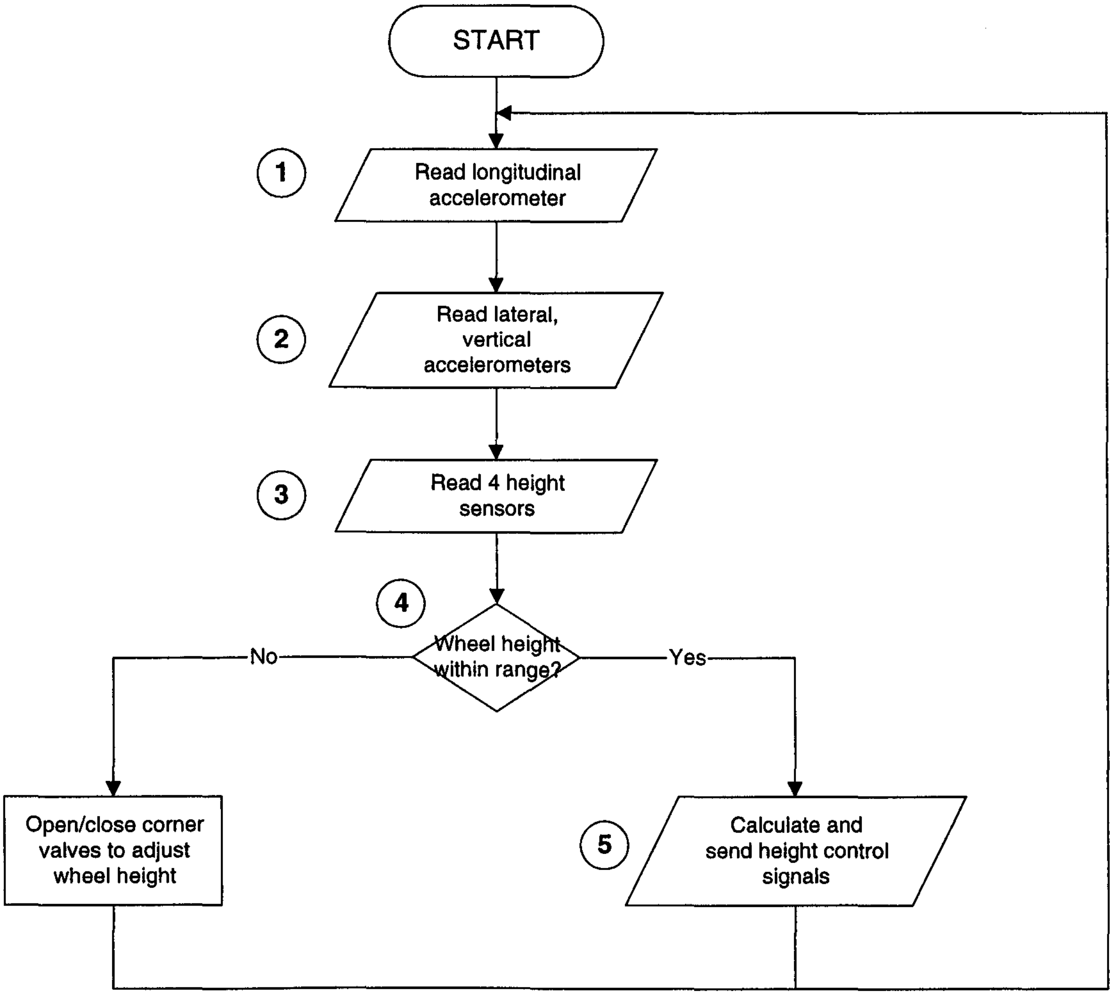
Asterisk(s) Explanation

****** 4WS allows steering direction change to settle before responding, in case there is a glitch in a steering angle sensor reading.

******* This { } is typical '*table look up*' program codes

Active Suspension

Active Suspension Flow Chart


START
1
Read longitudinal accelerometer
2
Read lateral, vertical accelerometers
3
Read 4 height sensors
4
Wheel height within range?
No
Yes
Open/close corner valves to adjust wheel height
5
Calculate and send height control signals

Active Suspension Pseudo Code

❶ Read longitudinal accelerometer

read longitudinal accelerometer (same as ABS ❶ but without division)

❷ Read lateral and vertical accelerometers

read lateral accelerometers (same as ABS ❶ but without division) x 2 (2 sensors)
read vertical accelerometers (same as ABS ❶ but without division) x 3 (3 sensors)

❸ Read wheel height sensors

read wheel height sensors (same as ABS ❶ but without division) x 4 (4 height sensors)

❹ Check if wheel heights within range

```
{
        load wheel height
        compare to lower limit
        compare to higher limit
}

{ } x 4 (4 wheels)
```

❺ Calculate and send wheel height control signals

wheel height control calculation according to the control equation below (described in exemplary active suspension system in Chapter 3)

wheel height demand = W +/- (Lo x K_{px}) +/- (La x K_{rx}) +/- (Ve x 1/(1+TS) x K_{bx})

W	- wheel height
Lo	- longitudinal acceleration
La	- lateral acceleration
Ve	- vertical acceleration
K_{px}, K_{rx}, K_{bx}	- relevant control gain
1/(1+TS)	- first-order delay element

Note that + or – sign depends on a particular wheel under calculation.

```
{
        load previous vertical acceleration
        load control gain
        multiply (Ve x 1/(1+TS) x $K_{bx}$)
        store result
}
```

{ } x 3 (same for longitudinal and lateral accelerations, but using current values i.e. no delay element)

```
load wheel height (W)
wheel height +/- longitudinal acc x control gain (W +/- (Lo x $K_{px}$))
result +/- lateral acc x control gain (W +/- (Lo x $K_{px}$) +/- (La x $K_{rx}$))
result +/- vertical acc x control gain (W +/- (Lo x $K_{px}$) +/- (La x $K_{rx}$) +/-
        (Ve x 1/(1+TS) x $K_{bx}$)
branch if +ve
        set valve control bit (open valve)
branch if –ve
        reset valve control bit (close valve)
```

Divisions (a/b)

```
Load b
Clear counter
Compare b to a
Branch if positive (b < a)
Increment counter
{accumulator content = b+b
  compare accumulator content to a
  branch if positive (quit the loop if accumulator content greater than a)
  increment counter } (repeat the { } loop until accumulator content greater than a)
{accumulator content – a (=c)
  compare c to b/2
  branch if negative
        increment counter} (round up the result)
( { } loop is to round the result according to the remainder)
store result
```

Assume the loop is repeated t times, instruction count is
1 x load
(t+2) x compare
(t+2) x branch
(t+1) x add/subtract
(t+2) x increment
1 x clear
1 x store

where t is the division result

Number of Instruction Execution Count

	Number of times an instruction is executed		
Microcontroller Instruction	**Active suspension**	**4WS**	**ABS**
Load	62	28	51
Branch	26	6	37
Compare	8	3	33
Store	32	12	26
Rotate	0	0	1
Clear	0	0	0
AND/OR/XOR	10	3	4
Increment	0	0	0
Set/reset bit	8	0	0
Add/subtract	12	3	10
Complement	0	0	0
Transfer (register <-> accumulator)	0	0	0
Clear carry	0	0	0
Multiply	12	5	4
Division	0	2	8

Table 1 Instruction execution count of active suspension, 4WS and ABS

Predicted Response Time of Alternative Architecture ECUs

The tables below contain the calculation process that leads to the response time prediction of alternative architecture ECUs, as described in Chapter 5.

Conventional Centralised, Conventional Centralised with Limited CAN Interaction, Conventional Centralised with Functional Integration Architectures (Architecture 1, 2, 4)

Active Suspension	Active Suspension Instruction Execution Count (n_x)		No. of Execution Cycles** per each instruction (c_x)			Active Suspension Total Execution Cycle Count (T_c)		
Microcontroller Instruction	MC without division capability (M68HC05)	MC with division capability (M68HC08&12)	M68HC05	M68HC08	M68HC12	M68HC05	M68HC08	M68HC12
Load	62	62	4	4	3	248	248	186
Branch	26	26	3	3	3	78	78	78
Compare	8	8	4	4	3	32	32	24
Store	32	32	5	4	3	160	128	96
Rotate	0	0	3	4	4	0	0	0
Clear	0	0	5	3	3	0	0	0
AND/OR/XOR	10	10	4	4	3	40	40	30
Increment	0	0	5	4	4	0	0	0
Set/reset bit	8	8	5	4	4	40	32	32
Add/subtract	12	12	4	4	3	48	48	36
Complement	0	0	3	1	1	0	0	0
Transfer (register <-> accumulator)	0	0	2	1	1	0	0	0
Clear carry	0	0	2	1	1	0	0	0
Multiply	12	12	11	5	3	132	60	36
Division	N/A	0	N/A	7	11	N/A	0	0
Sum						778	666	518
X multiplying factor (9)						7002	5994	4662
Total No. of execution cycles						7002	5994	4662
Cycle Time (ns)						476	125	125
control response time (ms)						3.3	0.7	0.6
Diagnostic overhead factor						1.6	1.6	1.6
response time (With diagnostics) ms						5.2	1.2	0.9
compiler overhead						1.3	1.3	1.3
response time (With diagnostics,compiler) ms						6.8	1.5	1.2
No. of interrupts (from signal table)***						12	12	12
Update rate (ms)						10.0	10.0	10.0
No. of interrupts per sec						1200	1200	1200
Percent of RTOS overhead						22.6	22.6	22.6
RTOS overhead (ms)						1.1	1.1	1.1
Predicted response time (ms)						7.9	2.7	2.3

4WS	4WS Instruction Execution Count (n_x)		No. of Execution Cycles** per each instruction (c_x)			4WS Total Execution Cycle Count (T_c)			
Microcontroller Instruction	MC without division capability (M68HC05)	MC with division capability (M68HC08&12)	M68HC05	M68HC08	M68HC12	M68HC05	M68HC05 (Extra for Division Instruction)*	M68HC08	M68HC12
Load	30	28	4	4	3	112	8	112	84
Branch	40	6	3	3	3	18	102	18	18
Compare	37	3	4	4	3	12	136	12	9
Store	12	12	5	4	3	60	0	48	36
Rotate	0	0	3	4	4	0	0	0	0
Clear	2	0	5	3	3	0	10	0	0
AND/OR/XOR	3	3	4	4	3	12	0	12	9
Increment	32	0	5	4	4	0	160	0	0
Set/reset bit	0	0	5	4	4	0	0	0	0
Add/subtract	35	3	4	4	3	12	128	12	9
Complement	2	0	3	1	1	0	6	0	0
Transfer (register <-> accumulator)	0	0	2	1	1	0	0	0	0
Clear carry	0	0	2	1	1	0	0	0	0
Multiply	5	5	11	5	3	55	0	25	15
Division	N/A	2	N/A	7	11	N/A	N/A	14	22
Sum						281	550	253	202
X multiplying factor (9)*						2529		2277	1818
Total No. of execution cycles						3079		2277	1818
Cycle Time (ns)						476		125	125
control response time (ms)						1.5		0.3	0.2
Diagnostic overhead factor						1.6		1.6	1.6
response time (With diagnostics) ms						2.3		0.4	0.4
compiler overhead						1.3		1.3	1.3
response time (With diagnostics,compiler) ms						3.0		0.6	0.5
No. of interrupts (from signal table)***						4		4	4
Update rate (ms)						10		10	10
No. of interrupts per sec						400		400	400
Percent of RTOS overhead						10.2		10.2	10.2
RTOS overhead (ms)						0.5		0.5	0.5
Predicted response time (ms)						3.5		1.1	1.0

ABS	**ABS Instruction Execution Count (n_x)**		**No. of Execution Cycles** per each instruction (c_x)**			**ABS Total Execution Cycle Count (T_c)**			
Microcontroller Instruction	MC without division capability (M68HC05)	MC with division capability (M68HC08&12)	M68HC05	M68HC08	M68HC12	M68HC05	M68HC05 (Extra for Division Instruction)*	M68HC08	M68HC12
Load	59	51	4	4	3	204	32	204	153
Branch	845	37	3	3	3	111	2424	111	111
Compare	841	33	4	4	3	132	3232	132	99
Store	26	26	5	4	3	130	0	104	78
Rotate	1	1	3	4	4	3	0	4	4
Clear	8	0	5	3	3	0	40	0	0
AND/OR/XOR	4	4	4	4	3	16	0	16	12
Increment	804	0	5	4	4	0	4020	0	0
Set/reset bit	0	0	5	4	4	0	0	0	0
Add/subtract	814	10	4	4	3	40	3216	40	30
Complement	8	0	3	1	1	0	24	0	0
Transfer (register <-> accumulator)	0	0	2	1	1	0	0	0	0
Clear carry	0	0	2	1	1	0	0	0	0
Multiply	4	4	11	5	3	44	0	20	12
Division		8	N/A	7	11	N/A	N/A	56	88
Sum						680	12988	687	587
X multiplying factor (9)*						6120		6183	5283
Total No. of execution cycles****						38216*		12366	10566
Cycle Time (ns)						476		125	125
control response time (ms)						9.1		0.8	0.7
Diagnostic overhead factor						1.6		1.6	1.6
response time (With diagnostics) ms						14.2		1.2	1.0
compiler overhead						1.3		1.3	1.3
response time (With diagnostics,compiler) ms						18.5		1.6	1.3
No. of interrupts (from signal table)***						4		4	4
Update rate (ms)						Proportional to wheel speed			
No. of interrupts per sec						3840*****		3840	3840
Percent of RTOS overhead						63.5		63.5	63.5
RTOS overhead (ms)						3.2		3.2	3.2
Predicted response time (ms)						21.7		**4.7**	4.5

* The division pseudo code is not multiplied by the correction factor, since it is believed to be in only the calculation parts identified, and not in the rest of the program.

** Extended addressing mode is assumed for each instruction, to allow more flexibility in memory allocation in programming

*** The number of input sensors counted here only includes those used in the control algorithms. Other sensors drawn in the wiring model in Chapter 6, such as oil level sensors, are not counted since they are read at much larger intervals and hence negligible.

**** The total response time of ABS is multiplied by 2 to make ABS response time limit (originally 5ms) consistent with those of 4WS and active suspension (10ms), for ease of comparison. This is only for comparison purpose and does not affect the actual system timing prediction.

***** A 100mph vehicle speed is assumed to obtain a high rate of interrupts, for worst case ECU response time prediction.

Total Centralised Architecture (Architecture 3)

No. of execution cycles	**M68HC05**	**M68HC08**	**M68HC12**
Active Suspension	7002	5994	4662
4WS	3079	2277	1818
ABS	38216	12366	10566
Total No. of execution cycles	48297	20637	17046
Cycle Time (ns)	476	125	125
control response time (ms)	23.0	2.6	2.1
Diagnostic overhead factor	1.6	1.6	1.6
response time (With diagnostics) ms	35.9	4.0	3.3
compiler overhead factor	1.3	1.3	1.3
response time (With diagnostics,compiler) ms	46.7	5.2	4.3
No. of interrupts per sec	5240	5240	5240
Percent of RTOS overhead	85.2	85.2	85.2
RTOS overhead (ms)	4.261	4.261	4.261
Predicted Response time	51.0	9.5	**8.6**

Distributed Architecture (Architecture 5)

The two tables below are the response time predictions of the centralised and distributed ECUs of the Distributed Architecture, respectively.

Central ECU	Centralised ECU Instruction Execution Count (n_x)		No. of Execution Cycles** per each instruction (c_x)			4WS Total Execution Cycle Count (T_c)			
Microcontroller Instruction	MC without division capability (M68HC05)	MC with division capability (M68HC08&12)	M68HC05	M68HC08	M68HC12	M68HC05	M68HC05 (Extra for Division Instruction)*	M68HC08	M68HC12
Load	28	26	4	4	3	104	8	104	78
Branch	41	7	3	3	3	21	102	21	21
Compare	38	4	4	4	3	16	136	16	12
Store	15	15	5	4	3	75	0	60	45
Rotate	2	2	3	4	4	6	0	8	8
Clear	2	0	5	3	3	0	10	0	0
AND/OR/XOR	3	3	4	4	3	12	0	12	9
Increment	32	0	5	4	4	0	160	0	0
Set/reset bit	12	12	5	4	4	60	0	48	48
Add/subtract	43	11	4	4	3	44	128	44	33
Complement	2	0	3	1	1	0	6	0	0
Transfer (register <-> accumulator)	1	1	2	1	1	2	0	1	1
Clear carry	0	0	2	1	1	0	0	0	0
Multiply	4	4	11	5	3	44	0	20	12
Division	N/A	2	N/A	7	11	N/A	N/A	14	22
Sum						384	550	348	289
X multiplying factor (9)*						3456		**3132**	**2601**
Total No. of execution cycles						4006		**3132**	**2601**
Cycle Time (ns)						476		125	125
control response time (ms)						1.91		0.39	0.33
Diagnostic overhead factor						1.6		1.6	1.6
response time (With diagnostics) ms						2.979		0.612	0.508
compiler overhead						1.3		1.3	1.3
response time (With diagnostics,compiler) ms						3.873		0.795	0.6604
No. of interrupts (from signal table)***						5		5	5
Update rate (ms)						10		10	10
No. of interrupts per sec						500		500	500
Percent of RTOS overhead						11.8		11.8	11.8
RTOS overhead (ms)						0.6		0.6	0.6
Predicted response time (ms)						**4.5**		1.4	1.2

Distributed ECU	Distributed ECU Instruction Execution Count (n_x)		No. of Execution Cycles** per each instruction (c_x)			ABS Total Execution Cycle Count (T_c)			
Microcontroller Instruction	MC without division capability (M68HC05)	MC with division capability (M68HC08&12)	M68HC05	M68HC08	M68HC12	M68HC05	M68HC05 (Extra for Division Instruction)*	M68HC08	M68HC12
Load	37	29	4	4	3	116	32	116	87
Branch	835	27	3	3	3	81	2424	81	81
Compare	810	2	4	4	3	8	3232	8	6
Store	10	10	5	4	3	50	0	40	30
Rotate	2	2	3	4	4	6	0	8	8
Clear	8	0	5	3	3	0	40	0	0
AND/OR/XOR	5	5	4	4	3	20	0	20	15
Increment	804	0	5	4	4	0	4020	0	0
Set/reset bit	22	22	5	4	4	110	0	88	88
Add/subtract	809	5	4	4	3	20	3216	20	15
Complement	8	0	3	1	1	0	24	0	0
Transfer (register <-> accumulator)	0	0	2	1	1	0	0	0	0
Clear carry	0	0	2	1	1	0	0	0	0
Multiply	5	5	11	5	3	55	0	25	15
Division	N/A	4	N/A	7	11	N/A	N/A	28	44
Sum						466	12988	434	389
X multiplying factor (9)*						4194		3906	3501
Total No. of execution cycles****						17182		3906	3501
Cycle Time (ns)						476		125	125
control response time (ms)						8.2		0.5	0.4
Diagnostic overhead factor						1.6		1.6	1.6
response time (With diagnostics) ms						12.8		0.8	0.7
compiler overhead						1.3		1.3	1.3
response time (With diagnostics,compiler) ms						16.6		1.0	0.9
No. of interrupts (from signal table)***						4		4	4
Update rate (ms)						10 except wheel speed sensor (proportional to wheel speed)			
No. of interrupts per sec						1260		1260	1260
Percent of RTOS overhead						23.5		23.5	23.5
RTOS overhead (ms)						1.2		1.2	1.2
Predicted response time (ms)						17.8		**2.2**	2.1

ROM ESTIMATION

ROM requirement is estimated based on method described in [2], which is developed by [3].

$N = k\,(0.5772 + \ln k)$
$L = N \div 3$
$ROM = L \times 20$

N	- estimated number of operators and operands in a program
k	- sum of estimated number of distinct operators in programming language and estimated number of distinct operands from the software design (sum of input, output, internal variables, constants and function modules)
L	- estimated number of program lines (assuming an average line has 3 operators and operands e.g. A = B)
ROM	- estimated ROM required (assuming each high level language program line is translated into 20 bytes of machine language code)

Operand Count

Active Suspension

Input and output variables (from wiring diagram)		**28**
Internal Variables (from Pseudo code)		**40**
Temporary variables for input and output readings	28	
A/D register, delay counter	2	
Previous vertical acceleration	1	
Product of acceleration and gain	3	
Sum of 3 acceleration x gain and wheel height	3	
Valve control register	1	
Status variables (oil level, pressure)	2	
Constants (from Pseudo code)		*13*
A/D completion constant	1	
Height range (min and max)	2	
Control gains	6	
Pressure range	2	
Oil level range	2	
Modules (from Flowchart and Pseudo code)		**13**
Read 3 accelerometers and height sensor	4	
Wheel height, oil level, pressure range check	3	
Wheel height adjust	1	

Gain x acceleration calculations	3
Wheel height calculation	1
Corner valve control	1

4WS

Input and output variables (from wiring diagram) *7*

Internal variables (from Pseudo code) *23*

7 temporary variables for input and output readings, previous steering angle, steering speed, settling time counter, 3 gain position index, 3 gains, 2 products of element and gain, 2-D table co-ordinate, actual 2-D table position memory address, A/D control register, A/D delay counter, direction change flag

Constants (from Pseudo code) *11*

reservoir tank range (2), step motor angle range (2), 3 table starting memory locations, 2 vehicle and steering speed intervals for 2-D table, total row in 2-D table, A/D completion constant

Modules (from Flowchart and Pseudo code) *15*

11 modules from flowcharts (except ❻), 3 table look up, rear wheel steering calculation

ABS

Input and output variables (from wiring diagram) *16*

Internal variables (from Pseudo code) *35*

16 temporary variables for input and output readings, A/D register, A/D counter, 4 wheel speeds, 2 diagonal wheel sums, vehicle speed, reference wheel speed, previous wheel speed, wheel acceleration, wheel slip, ABS status, current and next ABS states (2), 2 status (solenoid, motor), brake valve register

Constants (from Pseudo code) *11*

A/D completion constant, 5 state identifications, 3 acceleration thresholds (-a, a, +A), brake fluid range (2)

Modules (from Flowchart and Pseudo code) *25*

read wheel speed, scale wheel speed, 4 calculations (vehicle speed, wheel acceleration, slip, reference vehicle speed), ABS status check, 2 state determinations (current and next states), 15 state checks and controls (each state has acceleration and wheel slip checks, and brake valve set), brake fluid check

Distributed Architecture

Central ECU

Input and output variables (from wiring diagram) *34*

Internal variables (from Pseudo code) *63*

25 temporary variables for input and output readings, 16 4WS internal variables (except 7 I/O variables already included), A/D register, A/D delay counter, 4 status variables (active suspension oil level, pressure, ABS solenoid, motor), 4 wheel speeds, 2 diagonal wheel sums, vehicle speed

Constants (from Pseudo code) *18*

A/D completion constant, 6 limits (active suspension pressure, oil level, ABS brake fluid), 11 4WS constants

Modules (from Flowchart and Pseudo code) *20*

read longitudinal accelerometer, 3 checks (active suspension oil level, pressure, ABS oil level), 15 4WS modules, vehicle speed calculation

Distributed Wheel ECU

Input and output variables (from wiring diagram) *14*

Internal variables (from Pseudo code) *34*

14 temporary variables for input and output readings, A/D register, A/D delay counter, previous vertical acceleration, 3 products of acceleration and gain, Sum of 3 acceleration x gain and wheel height (3), valve control register, wheel speed, vehicle speed, reference wheel speed, previous wheel speed, wheel acceleration, wheel slip, ABS status, current and next ABS states (2), brake valve register

Constants (from Pseudo code) *17*

A/D completion constant, 2 wheel height limits, 6 control gains, 5 ABS state identifications, 3 acceleration thresholds

Modules (from Flowchart and Pseudo code) *33*

read vertical, lateral accelerometers and wheel height (3), wheel height check, wheel height adjust, 3 gain x acceleration calculations, wheel height calculation, read wheel speed, scale wheel speed, 3 calculations (wheel acceleration, slip, reference vehicle speed), ABS status check, 2 state determinations (current and next states), 15 ABS state checks and controls

The table below shows the ROM requirement calculation process.

ROM estimation	**Architectures 1,2, 4**			**Architecture 3**	**Architecture 5**	
	Active Suspension	*4WS*	*ABS*	*Total Centralised ECU*	*Central ECU*	*Distributed ECU*
I/Os	28	7	16	49	34	14
Internal variables	40	23	35	92	63	34
Constants	13	11	11	33	18	17
Functions (modules)	13	15	25	53	20	33
Total operands	94	56	87	227	135	98
Total operands with 50% extra	141	84	131	341	203	147
Total operators	53	53	53	53	53	53
k	194	137	184	394	256	200
N	1134	753	1062	2578	1564	1175
Program lines (L)	378	251	354	859	521	392
ROM estimate (bytes)	**7560**	**5021**	**7082**	**17189**	**10425**	**7834**

APPENDIX B VEHICLE SIGNAL MODELS FOR CAN SIMULATION

Table B1 is the list of signals used for CAN simulation of Conventional Centralised Architecture with Functional Integration Architecture (*Architecture 4*) in Chapter 5.

Signal No.	Name	Period (ms)	Class	Source ECU
1	spark output timing signal	5	C	PCM
2	front left wheel brake demand	5	C	ABS
3	front right wheel brake demand	5	C	ABS
4	rear left wheel brake demand	5	C	ABS
5	rear right wheel brake demand	5	C	ABS
6	ABS solenoid control	5	C	ABS
7	ABS motor control	5	C	ABS
8	front left wheel speed sensor	5	C	SEN
9	front right wheel speed sensor	5	C	SEN
10	rear left wheel speed sensor	5	C	SEN
11	rear right wheel speed sensor	5	C	SEN
12	crash sensor 1	5	C	PSS
13	crash sensor 2	5	C	PSS
14	crash sensor 3	5	C	PSS
15	brake position sensor	5	C	PSS
16	clutch position sensor	5	C	ICM
17	crankshaft position sensor	5	C	ICM
18	profile ignition pickup	5	C	ICM
19	speed control signal	5	C	CCS
20	transmission speed sensor	10	C	PCM
21	vehicle speed	10	C	ABS
22	front left wheel height demand	10	C	AS
23	front right wheel height demand	10	C	AS
24	rear left wheel height demand	10	C	AS
25	rear right wheel height demand	10	C	AS
26	fail-safe mode demand	10	C	AS
27	flow control valve control	10	C	AS
28	fan motor control	10	C	AS
29	variable timing pump control	10	C	AS
30	rear wheel steering demand	10	C	4WS
31	front left wheel height sensor	10	C	SEN
32	front right wheel height sensor	10	C	SEN
33	rear left wheel height sensor	10	C	SEN
34	rear right wheel height sensor	10	C	SEN
35	vertical acceleration sensor 1	10	C	SEN
36	vertical acceleration sensor 2	10	C	SEN

37	vertical acceleration sensor 3	10	C	SEN
38	lateral acceleration sensor 1	10	C	SEN
39	lateral acceleration sensor 2	10	C	SEN
40	longitudinal acceleration sensor	10	C	SEN
41	steering angle sensor	10	C	SEN
42	yaw rate sensor	10	C	SEN
43	rear wheel spin sensor 1	10	C	PSS
44	rear wheel spin sensor 2	10	C	PSS
45	cylinder Id sensor	20	B	PCM
46	manual level position	20	B	PCM
47	delta pressure feedback electronic	20	B	PCM
48	heated exhaust gas oxygen sensor	20	B	PCM
49	mass air flow sensor	20	B	PCM
50	throttle position sensor	20	B	PCM
51	engine RPM	20	B	PCM
52	ignition diagnostic monitor	20	B	ICM
53	transmission oil temperature	100	B	PCM
54	A/C compressor clutch	100	B	PCM
55	engine coolant temperature	100	B	PCM
56	transmission lubricant pressure	100	B	PCM
57	ABS solenoid status	100	B	SEN
58	ABS sensor status	100	B	SEN
59	4WS motor status	100	B	SEN
60	brake status	100	B	SEN
61	reverse switch	100	B	SEN
62	height switch	100	B	SEN
63	door switch	100	B	SEN
64	active suspension pressure status	100	B	SEN
65	battery current	100	B	ICM
66	battery voltage	100	B	ICM
67	shift sensor	100	B	ICM
68	intake air temperature	200	B	PCM
69	octane adjust plug	1s	A	PCM
70	transmission control switch	1s	A	PCM
71	engine idle speed	1s	A	PCM
72	engine status	1s	A	PCM
73	fuel flow	1s	A	PCM
74	transmission control indicator	1s	A	PCM
75	EGR vacuum regulator	1s	A	PCM
76	check engine indicator	1s	A	PCM
77	ABS warning lamp	1s	A	ABS
78	active suspension warning lamp	1s	A	AS
79	steering oil level	1s	A	SEN
80	suspension hydraulic oil level	1s	A	SEN
81	thermistor	1s	A	SEN
82	brake fluid	1s	A	SEN
83	power locks	1s	A	PSS

84	power seats	1s	A	PSS
85	power windows	1s	A	PSS
86	shift inhibit signal	1s	A	PSS
87	shift in progress	1s	A	PSS
88	seatbelt sensor	1s	A	PSS
89	door sensor 1	1s	A	PSS
90	door sensor 2	1s	A	PSS
91	door sensor 3	1s	A	PSS
92	door sensor 4	1s	A	PSS
93	door sensor 5	1s	A	PSS
94	anti-theft sensor	1s	A	PSS
95	airbag indicator lamp	1s	A	PSS
96	seatbelt lamp	1s	A	PSS
97	door lamps status	1s	A	PSS
98	airbag status	1s	A	PSS
99	fuel level sensor	1s	A	ICM
100	alternator warning indicator	1s	A	ICM
101	auto headlamp sensor	1s	A	ICD
102	ignition switch position	1s	A	ICD
103	horn sensor	1s	A	ICD
104	hazard sensor	1s	A	ICD
105	L/R signal	1s	A	ICD
106	control to tone maker	1s	A	ICD
107	oil pressure	1s	A	ICD
108	SET/ACCEL/RESUME	1s	A	CCS
109	cruise control indicator	1s	A	CCS
110	outside temperature	1s	A	CCM
111	desired temperature	1s	A	CCM
112	cabin temperature	1s	A	CCM
113	rear window defrost	1s	A	CCM
114	blower speed control	1s	A	CCM
115	damper control	1s	A	CCM
116	hear/cool control	1s	A	CCM
117	washer fluid sensor	10s	A	ICD

Table B1 Complete List of Signals for Conventional Centralised with Functional Integration CAN Simulation

Source ECU codes:

PCM	- Powertrain Control Module
ABS	- Antilock Braking System
AS	- Active Suspension
4WS	- 4 Wheel Steering
SEN	- Sensors
PSS	- Passenger Safety Systems
ICM	- Ignition Control Module
ICD	- Instrument Cluster Display
CCS	- Cruise Control System
CCM	- Climate Control Module

Table B2 is the list of signals, which are different from the above Table B1, used for CAN simulation of Distributed Wheel Architecture (*Architecture 5*) in Chapter 5.

Signal No.	**Name**	**Period (ms)**	**Class**	**Source ECU**
2	front left wheel brake demand	5	C	FLC
3	front right wheel brake demand	5	C	FRC
4	rear left wheel brake demand	5	C	RLC
5	rear right wheel brake demand	5	C	RRC
6	ABS solenoid control	5	C	DCC
7	ABS motor control	5	C	DCC
8	front left wheel speed	5	C	FLC
9	front right wheel speed	5	C	FRC
10	rear left wheel speed	5	C	RLC
11	rear right wheel speed	5	C	RRC
21	vehicle speed	10	C	DDC
22	front left wheel height demand	10	C	FLC
23	front right wheel height demand	10	C	FRC
24	rear left wheel height demand	10	C	RLC
25	rear right wheel height demand	10	C	RRC
26	fail-safe mode demand	10	C	DCC
27	flow control valve control	10	C	DCC
28	fan motor control	10	C	DCC
29	variable timing pump control	10	C	DCC
30	rear wheel steering demand	10	C	DCC
31	front left wheel height	10	C	FLC
32	front right wheel height	10	C	FRC
33	rear left wheel height	10	C	RLC
34	rear right wheel height	10	C	RRC
77	ABS warning lamp	1s	A	DCC
78	active suspension warning lamp	1s	A	DCC

Table B2 Changes in Signals from Table B1 for Distributed Wheel Architecture CAN Simulation

ECU codes:
FLC - Front Left Wheel Controller
FRC - Front Right Wheel Controller
RLC - Rear Left Wheel Controller
RRC - Rear Right Wheel Controller
DCC - Distributed Central Controller

The figure in the next page is the CAN simulation model on *Simul8* software.

APPENDIX C IN-VEHICLE SIGNAL LIST FOR VEHICLE WIRING MODELS

Signal Name	Source / Destination (Component no.)*
Engine Management System (EMS) (Inputs)	
throttle position	driver controls (1)
ignition key status	driver controls (2)
gearbox status	transmission
Engine Management System (Outputs)	
malfunction display	driver display
Transmission Control (Inputs)	
throttle demand	driver controls (1)
throttle angle	EMS
drive style selection	driver switches
kick down	driver controls (10)
gear lever position	driver controls (11)
engine speed	EMS
engine torque	EMS
vehicle speed	vehicle speed sensor (7)
Transmission Control (Outputs)	
engine torque demand	EMS
malfunction display	driver display
Cruise Control (Inputs)	
vehicle speed	vehicle speed sensor (7)
on/off selection	driver switches
resume acceleration	driver switches
brake lamp	driver controls (14)
throttle status	driver controls (1)
engine speed	EMS
Cruise Control (Outputs)	
throttle demand	throttle actuator (O2)
cruise status	driver display
ABS/ASR (Inputs)	
wheel speed (x4)	wheel speed sensors (9)
engine speed	EMS

main throttle angle	EMS
sub-throttle angle	EMS
traction motor current	traction hydraulic system
main throttle idle	EMS
sub throttle idle	EMS
accumulator pressure	traction hydraulic system
traction on/off	driver switches
brake fluid level	brake fluid level switch (12)
parking brake status	driver controls (13)
engine system status	EMS
gear shift position	driver controls (11)
stop light	driver controls (14)
step motor power supply	traction hydraulic system
ABS solenoid status	ABS hydraulic system
ABS motor status	ABS hydraulic system
ABS/ASR (Outputs)	
ABS wheel brake demand (x4)	ABS wheel actuator (O6)
traction brake actuator demand (x3)	traction hydraulic system
sub-throttle demand	sub-throttle actuator (O3)
ABS/traction throttle demand	EMS
ABS motor relay control	ABS hydraulic system
ABS solenoid relay control	ABS hydraulic system
traction relay and motor controls (x3)	traction hydraulic system
ABS indicator lamp	driver display
traction indicator lamp	driver display
traction on/off lamp	driver display
traction status	EMS
Active Suspension Control (Inputs)	
vertical acceleration (x3)	vertical G sensors (3)
lateral acceleration (x2)	lateral G sensors (4)
vehicle height (x4)	height sensors (5)
longitudinal acceleration	longitudinal G sensor (6)
vehicle speed	vehicle speed sensor (7)
engine speed	EMS
brake signal	driver controls (14)
door status	door switch (8)
vehicle height demand	driver switches
parking brake status	driver controls (13)
pressure status	suspension hydraulic system
thermistor	suspension hydraulic system
oil level status	suspension hydraulic system
Active Suspension Control (Outputs)	

individual wheel height demands (x4)	suspension wheel pressure control valves (O1)
fail-safe mode display	driver display
fail-safe mode demand	suspension hydraulic system
variable pump control	suspension hydraulic system
flow control valve control	suspension hydraulic system
fan motor control	suspension hydraulic system
Power Steering (Inputs)	
steering wheel angular velocity	steering wheel angular velocity sensor (15)
steering wheel torque	steering wheel torque sensor (16)
vehicle speed	vehicle speed sensor (7)
Power Steering (Outputs)	
power demand	power steering unit (5)
malfunction display	driver display
4 Wheel Steering System (Inputs)	
vehicle speed	vehicle speed sensor (7)
steering wheel angle	steering wheel angle sensors (17)
hydraulic oil level switch	4WS hydraulic oil level sensor (19)
reverse switch	driver controls (11)
yaw rate	yaw rate sensor (18)
motor rotating angle	4WS actuator
4 Wheel Steering (Outputs)	
rear wheel steer demand	4WS hydraulic system (O4)

Table C1 Signals within vehicle electronic wiring models

* Source/Component numbers are designated to identify them in the wiring diagrams in Chapter 6.

APPENDIX D SYSTEM COMPONENT COST CALCULATION

Here are the averaged component price list used in the calculation in Chapter 6 [5]:

- IC price is the mean of average Op-Amp IC price of £0.80 and average logic IC price of £0.19, and is equal to £0.49.
- Resistor price is an average price of metal thin film resistor, and is equal to £0.10
- Capacitor price is an average price of multi-layer ceramic through hole capacitors, and is equal to £0.19
- Transistor price is an average price of BJT and FET transistors, and is equal to £1.06

The prices are taken of the types of ICs, resistors and capacitors that are most often found on the powertrain and dynamic control ECUs, which the author processes. Due to a variety of transistors found, the prices of all types of transistors from the catalogue are taken.

APPENDIX E ALTERNATIVE ARCHITECTURE RELIAIBILITY MODELLING

The reliability predicting equations based on the prediction technique using automotive data [6], described in Chapter 7, are written below:

For integrated circuits:	$\lambda_p = \lambda_b.\pi_F.\pi_S.\pi_P.\pi_T$
For resistors:	$\lambda_p = \lambda_b.\pi_L.\pi_S.\pi_P.\pi_T$
For capacitors:	$\lambda_p = \lambda_b.\pi_L.\pi_P.\pi_T$
For transistors:	$\lambda_p = \lambda_b.\pi_L.\pi_S.\pi_T$

Where:

λ_p	- predicted component failure rate
λ_b	- base failure rate
π_F	- integrated circuit family
π_L	- location factor
π_S	- screening factor
π_P	- module packaging factor
π_T	- temperature factor

The reliability calculation based on the reliability prediction technique developed by [6] is shown in Table E1.

	No. of components	λ_b	π_F	π_L	$\pi_S^{\#}$	$\pi_P^{\#\#}$	$\pi_T^{\#\#\#}$	λ_p
Architecture 1								
Microcontrollers	3	0.0038	1	N/A	1	1	1.36	0.016
ICs*	15	0.0038	1	N/A	1	1	1.2	0.068
Resistors**	194	0.000041	N/A	1	1	1	1	0.008
Capacitors***	96	0.00057	N/A	1	N/A	1	1	0.055
Transistors****	51	0.00415	N/A	1	1	N/A	1.17	0.248
Total λ_p								**0.394**
Architecture 2,4								
Microcontrollers	3	0.0038	1	N/A	1	1	1.36	0.016
ICs	12	0.0038	1	N/A	1	1	1.2	0.055
Resistors	168	0.000041	N/A	1	1	1	1	0.007
Capacitors	79	0.00057	N/A	1	N/A	1	1	0.045
Transistors	51	0.00415	N/A	1	1	N/A	1.17	0.248
Total λ_p								**0.370**
Architecture 3								
Microcontrollers	1	0.0038	1	N/A	1	1	1.36	0.005
ICs	27	0.0038	1	N/A	1	1	1.2	0.123
Resistors	240	0.000041	N/A	1	1	1	1	0.010
Capacitors	179	0.00057	N/A	1	N/A	1	1	0.102
Transistors	19	0.00415	N/A	1	1	N/A	1.17	0.092
Total λ_p								**0.332**
Architecture 5 (Centralised ECU)								
Microcontrollers	1	0.0038	1	N/A	1	1	1.36	0.005
ICs	11	0.0038	1	N/A	1	1	1.2	0.050
Resistors	112	0.000041	N/A	1	1	1	1	0.005
Capacitors	64	0.00057	N/A	1	N/A	1	1	0.036
Transistors	18	0.00415	N/A	1	1	N/A	1.17	0.087
Total λ_p								**0.184**
Architecture 5 (Distributed ECUs)								
Microcontrollers	4	0.0038	1	N/A	1	1	2.5	0.038
ICs	4	0.0038	1	N/A	1	1	1.94	0.029
Resistors	120	0.000041	N/A	1	1	1	1.01	0.005
Capacitors	44	0.00057	N/A	1	N/A	1	1.05	0.026
Transistors	60	0.00415	N/A	1	1	N/A	1.24	0.309
Total λ_p								**0.408**

#, ## Since the ECUs here are all highly safety related, the most reliable screening (electrical and environmental) and packaging (encapsulated) on the list are assumed.

No data on passenger compartment location (which most ECUs are, except Distributed ECUs, which are chassis mounted), so data on trunk location is used instead. This is because trunk location offers the least hostile place in the list, which is believed to be closed to passenger compartment.

* MOS type ** metal film type *** nonelectrolytic type **** average of FET and BJT values since they are approximately equal in numbers

Table E1 Alternative architecture reliability calculation based on automotive data

References

1. *Jurgen R* **Automotive Electronics Handbook 2nd edition** McGraw-Hill, 1999
2. *Lawrence P, et al.* **Real-Time Microcomputer System Design: An Introduction** McGraw-Hill, 1987
3. *Shooman M* **Software Engineering: Design, Reliability, and Management** McGraw-Hill, 1983
4. *Kernighan B, et al.* **The C Programming Language 2nd edition** Prentice Hall, 1988
5. **RS Catalogue** RS Components Ltd., 2000
6. *Denson W, et al.* **Automotive Electronic Reliability Prediction** SAE No.870050