



Waterford Institute of Technology

Institiúid Teicneolaíochta Phort Láirge

**Improving the Reliability and Performance of
FlexRay Vehicle Network Applications Using
Simulation Techniques**

Robert Shaw B.Sc. (Hons)

M.Sc.

Supervisor: Brendan Jackman B.Sc., M.Tech.

Submitted to the Waterford Institute of Technology

Awards Council, 27 May 2009.

ACKNOWLEDGEMENTS

Acknowledgements

I would like to thank the following people for all their support and help over the past two years of this project. Without them this thesis would not be possible.

Firstly I would like to thank Mr. Brendan Jackman for all his encouragement, guidance and enthusiasm throughout this project.

I would also like to thank my fellow group members during the time span of this project, for all their help, guidance and advice:

- Frank Walsh, Group Supervisor, Department of Computing, Maths and Physics, Waterford Institute of Technology
- David Power, Group Supervisor, Department of Computing, Maths and Physics, Waterford Institute of Technology
- Gareth Leppla, Group Member, Department of Computing, Maths and Physics, Waterford Institute of Technology
- Richard Murphy, Group Member, Department of Computing, Maths and Physics, Waterford Institute of Technology
- John Walsh, Group Member, Department of Computing, Maths and Physics, Waterford Institute of Technology

I would also like to thank my friends for all their support and patience while I undertook this project.

Special thanks go to Gillian Chester who showed great patience, support and encouragement throughout this project.

I would lastly and most importantly like to thank my parents who were more of a help than they will ever know.

DECLARATION

Declaration

I, Robert Shaw, declare that this thesis is submitted by me in partial fulfilment of the requirement for the degree M.Sc., is entirely my own work except where otherwise accredited. It has not at any time either whole or in part been submitted for any other educational award.

Signature: _____

Robert Shaw,
27 May 2009.

Abstract

Modern vehicles are becoming more and more sophisticated, with more functions being controlled by a microprocessor unit. As new functions are developed there is not only more of a demand on the control unit, but there is also more demand placed on the communication network(s) within a car. There is also a growing need for fast and dependable networks for new safety features such as X-by-wire applications.

A trend in the automotive industry to make cars more eco-friendly has emerged. As the amount of applications increases then the number of wires within a car increases and this can potentially add a large amount of weight leading to increased fuel consumption. This along with the need for higher performance networks led to the development of the FlexRay protocol.

FlexRay is a newly developed network protocol that is intended to address the current and future needs of the automotive industry. It is backed by many automotive manufacturers and suppliers. As such, FlexRay looks increasingly likely to become the network application of choice for many companies, especially where safety critical systems are implemented.

The purpose of this research was to design, implement and test a simulation model of a FlexRay network node. This simulation model could be a benefit to system developers to ensure accurate communication is achieved by tracing the flow of information through a FlexRay-based system and ensuring all timing constraints are met. The model was built using MATLAB, Simulink and SimEvents. The basis of the model was a node that incorporated a separate host microcontroller and communications controller. The communications controller was based on The Bosch E-Ray IP. The simulation model comprised of the application, software driver, communications and physical bus layers of a FlexRay based system. The model was then calibrated against a real world system over a number of different test cases and constraints.

The final part of the research involved running tests to determine if the model that has been built, was built in a correct manner i.e. validation of the simulation model. The model was then evaluated for its ability to carry out its intended role.

TABLE OF CONTENTS

Table of Contents

<i>Acknowledgements</i>	<i>ii</i>
<i>Declaration</i>	<i>iii</i>
<i>Abstract</i>	<i>iv</i>
<i>Table of Contents</i>	<i>v</i>
<i>Table of Figures</i>	<i>x</i>
<i>Table of Tables</i>	<i>xvi</i>
SECTION I: THESIS OVERVIEW	1
CHAPTER 1 . THESIS OVERVIEW	2
1.1 Problem Specification.....	2
1.2 Research Questions	2
1.3 Document Layout.....	3
1.4 References.....	5
SECTION II: LITERARY REVIEW	6
CHAPTER 2 . LITERARY REVIEW INTRODUCTION.....	7
2.1 Scope.....	7
2.2 Terminology.....	8
2.3 Criteria for Discussion	9
2.4 Limits of the Review.....	9
CHAPTER 3 . AUTOMOTIVE NETWORKS.....	10
3.1 Introduction	10
3.2 Automotive Networks.....	10
3.3 Networking Type Overview.....	12
3.4 Automotive Network Protocols	14
3.5 Event-Triggered Protocols	16
3.6 Time-Triggered Protocols	23
3.7 Automotive Network Design	30
3.8 Conclusion.....	38
3.9 References.....	38

TABLE OF CONTENTS

CHAPTER 4 . FLEXRAY	45
4.1 Introduction	45
4.2 Network Topology.....	46
4.3 FlexRay Hardware.....	47
4.4 Global Time and Timing.....	48
4.5 Media Access Control.....	51
4.6 Frame Format.....	53
4.7 Coding & Decoding.....	56
4.8 Wakeup	62
4.9 Conclusion.....	64
4.10 References.....	64
CHAPTER 5 . PERFORMANCE ANALYSIS	66
5.1 Introduction	66
5.2 System Performance and Analysis.....	66
5.3 Software Metrics.....	73
5.4 Previous Systems Analysis.....	80
5.5 Conclusion.....	103
5.6 References.....	104
CHAPTER 6 . E-RAY	107
6.1 Introduction	107
6.2 Features	107
6.3 Components	108
6.4 Register Map.....	110
6.5 Communication Controller States	112
6.6 Error Handling.....	116
6.7 Message Handling	120
6.8 Message RAM	124
6.9 Filtering and Masking	129
6.10 FIFO.....	131
6.11 Packaging	133
6.12 Conclusion.....	134
6.13. References.....	134
CHAPTER 7 . DISCRETE EVENT SIMULATION	136
7.1 Introduction	136
7.2 Systems	137
7.3 Simulation Process	139

TABLE OF CONTENTS

7.4 Building Models.....	140
7.5 Validation & Verification	144
7.6 Tests and Analysis.....	145
7.7 Simulation of Queues, Statistics and Random Numbers	146
7.8 Simulation Software.....	150
7.9 MATLAB, Simulink and SimEvents.....	152
7.10 Simulation Software Selection	162
7.11 Conclusion.....	165
7.11. References.....	166
CHAPTER 8 . FLEXRAY SOFTWARE DRIVERS.....	170
8.1 Introduction	170
8.2 COMMSTACK.....	171
8.3 AUTOSAR.....	174
8.4 Fujitsu FlexRay Driver.....	179
8.5 Conclusion.....	182
8.6 References.....	182
CHAPTER 9 . LITERARY REVIEW CONCLUSION	185
9.1 Literary Review Summary.....	185
9.2 Available Literature.....	186
9.3 Areas of Further Study.....	186
SECTION III: MODEL DEVELOPMENT.....	187
CHAPTER 10 . METHODOLOGY	188
10.1 Introduction	188
10.2 Simulation Process	188
10.3 Simulation Process in Relation to the Research.....	192
10.4 Conclusion.....	193
10.6 References.....	194
CHAPTER 11 . SIMULATION MODEL DEVELOPMENT.....	195
11.1 Introduction	195
11.2 Specification Development Process.....	197
11.3 Simulation Model Specifications	198
11.4 Model Metrics.....	207
11.5 The Model.....	212

TABLE OF CONTENTS

<i>11.6 Conclusion</i>	238
<i>11.7 References</i>	238
CHAPTER 12 . VERIFICATION	240
<i>12.1 Introduction</i>	240
<i>12.2 Verification</i>	241
<i>12.3 Model Subsystem Debugging</i>	243
<i>12.4 Simulation Model Verification</i>	247
<i>12.5 Model Execution Time</i>	257
<i>12.6 Conclusion</i>	259
<i>12.7 References</i>	260
SECTION IV: MODEL CALIBRATION & VALIDATION	261
CHAPTER 13 . CALIBRATION	262
<i>13.1 Introduction</i>	262
<i>13.2 Test Equipment</i>	264
<i>13.3 Calibration Procedure</i>	275
<i>13.4 Calibration Test Cases</i>	285
<i>13.5 Calibration Data & Results</i>	287
<i>13.6 Conclusion</i>	320
<i>13.7 References</i>	321
CHAPTER 14 . VALIDATION	323
<i>14.1 Introduction</i>	323
<i>14.2 Validation Procedure</i>	324
<i>14.3 Validation Data Collection</i>	326
<i>14.4 Validation Review</i>	328
<i>14.5 Conclusion</i>	343
<i>14.6 References</i>	345
SECTION V: CONCLUSION	346
CHAPTER 15 . CONCLUSION	347
<i>15.1 Introduction</i>	347
<i>15.2 Research Summary</i>	347
<i>15.3 Research Questions</i>	348

TABLE OF CONTENTS

<i>15.4 Research Conclusions</i>	350
<i>15.5 Area of Further Study</i>	353
<i>15.6 References</i>	354
SECTION VI: BIBLIOGRAPHY	356
SECTION VII: APPENDICES	I
APPENDIX A: MODEL COLOUR CODING	II
APPENDIX B: MODEL VARIABLES & ATTRIBUTES	V
APPENDIX C: THE MODEL	XVI
<i>Source Code</i>	<i>XVII</i>
APPENDIX D: TECHNICAL PAPERS	XVIII
<i>2008 IEEE International Symposium on Industrial Electronics, 30 June - 2 July 2008, Cambridge, United Kingdom 'An Introduction to FlexRay as an Industrial Network' Robert Shaw, Brendan Jackman</i>	<i>XIX</i>

TABLE OF FIGURES

Table of Figures

FIGURE 3.1: COMPUTER COMPONENTS AND POSSIBLE CONNECTIONS	11
FIGURE 3.2: EVENT-TRIGGERED AND TIME-TRIGGERED NETWORK PATTERNS.....	13
FIGURE 3.3: AUTOMOTIVE NETWORK APPLICATIONS	15
FIGURE 3.4: AUTOMOTIVE NETWORKS FUNCTIONALITY BREAKDOWN	15
FIGURE 3.5: CAN BUS	17
FIGURE 3.6: CAN BUS ARBITRATION	18
FIGURE 3.7: CAN STANDARD FRAME FORMAT.....	18
FIGURE 3.8: LIN BUS WITH SINGLE MASTER NODE AND ‘N’ SLAVE NODES	19
FIGURE 3.9: LIN COMMUNICATION.....	20
FIGURE 3.10: MOST25 FRAME	22
FIGURE 3.11: MOST25 FRAME	23
FIGURE 3.12: TTP BUS TOPOLOGY	24
FIGURE 3.13: TTP COMMUNICATION CYCLE	25
FIGURE 3.14: FLEXRAY FRAME.....	26
FIGURE 3.15: FLEXRAY HYBRID TOPOLOGY.....	26
FIGURE 3.16: FLEXRAY BUS ACCESS.....	27
FIGURE 3.17: CYCLE MULTIPLEXING.....	28
FIGURE 3.18: RATE DIFFERENCES	29
FIGURE 3.19: OFFSET DIFFERENCES	30
FIGURE 3.20: RATE AND OFFSET CORRECTION APPLIED	30
FIGURE 3.21: DA VINCI FLEXRAY SCHEDULE DESIGN	32
FIGURE 3.22: EB TRESOS PRODUCT FAMILY	33
FIGURE 4.1: A PASSIVE BUS TOPOLOGY.....	46
FIGURE 4.2: SINGLE CHANNEL HYBRID NETWORK.....	46
FIGURE 4.3: LOGICAL INTERFACE	47
FIGURE 4.4: TIMING HIERARCHY.....	48
FIGURE 4.5: THE RELATIONSHIP BETWEEN CLOCK SYNCHRONISATION AND THE MEDIA ACCESS TIME FRAME	50
FIGURE 4.6: COMMUNICATION CYCLE.....	51
FIGURE 4.7: FRAME FORMAT.....	53
FIGURE 4.8: ENCODED BIT STREAM.....	56
FIGURE 4.9: RECEIVED BIT STREAM	58
FIGURE 4.10: CAS AND MTS ENCODING	59
FIGURE 4.11: WAKEUP PATTERN USING TWO WAKEUP SYMBOLS.....	60
FIGURE 4.12: SAMPLING AND MAJORITY VOTING OF A RECEIVED BIT PATTERN AT THE INPUT	62

TABLE OF FIGURES

FIGURE 5.1: INTERRUPT-DRIVEN RESPONSE TIME.....	68
FIGURE 5.2: TIMING CHART EXAMPLE.....	70
FIGURE 5.3: STANDARD MEMORY MAP	71
FIGURE 5.4: THE EFFECT OF BAD SOFTWARE ON A COMPANY.....	75
FIGURE 5.5: NUMBER OF FAULTS FOUND IN SOFTWARE	76
FIGURE 5.6: FLEXRAY CONCEPTUAL ARCHITECTURE.....	81
FIGURE 5.7: SCHEDULED TRANSMISSION DEFINITION.....	82
FIGURE 5.8: FLEXRAY MODEL SYSTEM BASE	84
FIGURE 5.9: SYSTEM DEVELOPMENT PROCESS.	85
FIGURE 5.10: THE SMART DESIGNER WORKFLOW.....	86
FIGURE 5.11: THE SMART SIMULATOR ARCHITECTURE	87
FIGURE 5.12: SMARTOSEK ENGINE CONTROL SYSTEM.....	88
FIGURE 5.13: EVENT STREAM WITH P=4 AND J=1	89
FIGURE 5.14: SYMTAS DEVELOPED MODEL.....	90
FIGURE 5.15: SCHEDULING ALGORITHM.....	91
FIGURE 5.16: NETWORK GATEWAY SIMULINK/SIMEVENTS MODEL.....	93
FIGURE 5.17: AUTOMESH ARCHITECTURE.....	94
FIGURE 5.18: ABSTRACT DISTRIBUTED SYSTEM.....	96
FIGURE 5.19: CO-SIMULATION MAPPED EXAMPLE.....	96
FIGURE 5.20: SIMULATOR COMMUNICATION STRUCTURE	97
FIGURE 5.21: NS-2 MODEL SHOWING THE INTERFACE MODULE	98
FIGURE 5.22: ROUTER BLOCK MODEL	99
FIGURE 6.1: BLOCK DIAGRAM OF THE WORKINGS OF AN E-RAY CHIP	108
FIGURE 6.2: POSSIBLE COMMUNICATIONS CONTROLLER STATES	112
FIGURE 6.3: STATE DIAGRAM FOR NODE ENTERING STARTUP.....	114
FIGURE 6.4: RAM BLOCKS WITH LOCAL PARITY GENERATORS AND CHECKERS	119
FIGURE 6.5: HOST – MESSAGE RAM INTERFACE.....	121
FIGURE 6.6: DOUBLE BUFFER STRUCTURE INPUT	122
FIGURE 6.7: SWAPPING INPUT BUFFER COMMAND MASK & INPUT BUFFER COMMAND REGISTER BITS.....	122
FIGURE 6.8: DOUBLE BUFFER STRUCTURE OUTPUT	123
FIGURE 6.9: SWAPPING OUTPUT BUFFER COMMAND MASK & OUTPUT BUFFER COMMAND REGISTER BITS	123
FIGURE 6.10: TRANSIENT BUFFER RAMS	124
FIGURE 6.11: MESSAGE RAM CONFIGURATION EXAMPLE.....	125
FIGURE 6.12: HEADER SEGMENT IN MESSAGE RAM.....	126
FIGURE 6.13: DATA PARTITION IN MESSAGE RAM EXAMPLE.....	128
FIGURE 6.14: EMPTY, NOT EMPTY AND OVERRUN STATES.....	132
FIGURE 7.1: DISCRETE-SYSTEM STATE VARIABLE.....	138
FIGURE 7.2: CONTINUOUS-SYSTEM STATE VARIABLE	138

TABLE OF FIGURES

FIGURE 7.3: SIMULATION STUDY STEPS	140
FIGURE 7.4: SIMULATION MODELS	142
FIGURE 7.5: FLIGHT CONTROL HILS SYSTEM.....	143
FIGURE 7.6: MODEL BUILDING, VERIFICATION AND VALIDATION.....	145
FIGURE 7.7: OPEN QUEUING NETWORK	149
FIGURE 7.8: CLOSED QUEUING NETWORK	149
FIGURE 7.9: MATHWORKS PRODUCT OVERVIEW	153
FIGURE 7.10: FIRST MATLAB GRAPHICS.....	154
FIGURE 7.11: MODERN MATLAB GRAPH	154
FIGURE 7.12: MATLAB GRAPHICAL DEVELOPMENT	155
FIGURE 7.13: MATLAB ENVIRONMENT	155
FIGURE 7.14: AN M-FILE	156
FIGURE 7.15: SIMULINK ENVIRONMENT	157
FIGURE 7.16: SIMULINK LIBRARY	157
FIGURE 7.17: SIMEVENTS LIBRARY WINDOW.....	158
FIGURE 7.18: SIMEVENTS ENTITY GENERATOR OBJECTS	159
FIGURE 7.19: SIMEVENTS TUTORIAL BLOCKS	160
FIGURE 7.20: SINGLE SERVER PARAMETERS BOX.....	161
FIGURE 7.21: TUTORIAL BLOCKS CONNECTED	161
FIGURE 7.22: TUTORIAL RESULTS	162
FIGURE 8.1: FLEXRAY SOFTWARE DRIVER OPTIONS.....	171
FIGURE 8.2: COMMSTACK SYSTEM OVERVIEW	172
FIGURE 8.3: COMMSTACK SYSTEM ARCHITECTURE	173
FIGURE 8.4: COMMSTACK STATE DIAGRAM.....	174
FIGURE 8.5: VIRTUAL FUNCTIONAL BUS CONCEPT	176
FIGURE 8.6: SOFTWARE COMPONENT COMMUNICATION INTERFACE TYPES.....	177
FIGURE 8.7: FLEXRAY STACK LAYOUT	178
FIGURE 8.8: FUJITSU FLEXRAY DRIVER LAYERS	180
FIGURE 8.9: FUJITSU FLEXRAY DRIVER ARCHITECTURE	180
FIGURE 8.10: FUJITSU FLEXRAY DRIVER SERVICES	181
FIGURE 10.1: SIMULATION STUDY STEPS	189
FIGURE 11.1: FLEXRAY DEVELOPMENT STEPS	196
FIGURE 11.2: FLEXRAY NODE ELEMENTS	197
FIGURE 11.3: APPLICATION INPUTS, OUTPUTS AND CONSIDERATIONS.....	198
FIGURE 11.4: SOFTWARE DRIVER INPUTS, OUTPUTS AND CONSIDERATIONS.....	199
FIGURE 11.5: COMMUNICATIONS CONTROLLER INPUTS, OUTPUTS AND CONSIDERATIONS	199
FIGURE 11.6: PHYSICAL BUS INPUTS, OUTPUTS AND CONSIDERATIONS	200

TABLE OF FIGURES

FIGURE 11.7: TOP LAYER OF SIMULATION MODEL.....	201
FIGURE 11.8: FLEXRAY MODEL SUBSECTIONS	201
FIGURE 11.9: ENTITY PATHS.....	202
FIGURE 11.10: SLOT ENTITY PATHS.....	204
FIGURE 11.11: REQUEST ENTITY PATHS	204
FIGURE 11.12: FRAME ENTITY PATHS.....	205
FIGURE 11.13: CYCLE ENTITY PATHS	205
FIGURE 11.14: MODEL AS A FLOW OF DATA.....	207
FIGURE 11.15: E-RAY DATA FLOW PATH.....	208
FIGURE 11.16: E-RAY BLOCK DIAGRAM	213
FIGURE 11.17: COMMUNICATIONS CONTROLLER TASKS	213
FIGURE 11.18: MODEL OF THE COMMUNICATIONS CONTROLLER	214
FIGURE 11.19 TRANSIENT BUFFER RAM STRUCTURE	215
FIGURE 11.20: SYNCHRONISATION BLOCK.....	216
FIGURE 11.21: CYCLE ENTITY FLOW DIAGRAM.....	217
FIGURE 11.22: INITIALISATION BLOCK.....	218
FIGURE 11.23: INITIALISE STATIC SEGMENT BLOCK	218
FIGURE 11.24: STATIC SEGMENT BLOCK	220
FIGURE 11.25: GET START OF CYCLE	220
FIGURE 11.26: DYNAMIC SEGMENT BLOCK.....	221
FIGURE 11.27: DYNAMIC CHANNEL BLOCK.....	221
FIGURE 11.28: DYNAMIC ENABLE BLOCK	221
FIGURE 11.29: NETWORK IDLE TIME AND SYMBOL WINDOW BLOCK.....	222
FIGURE 11.30: INCREMENT CYCLE COUNT BLOCK.....	223
FIGURE 11.31: GLOBAL TIME UNIT.....	224
FIGURE 11.32: CYCLE COUNT ATTRIBUTE ADDER	224
FIGURE 11.33: DYNAMIC SLOT GENERATOR.....	225
FIGURE 11.34: MESSAGE HANDLER	226
FIGURE 11.35: MESSAGE RAM MODEL OPERATION	227
FIGURE 11.36: MESSAGE RAM BUFFERS	227
FIGURE 11.37: MESSAGE HANDLER MODEL BLOCKS	228
FIGURE 11.38: OUTPUT BUFFER STRUCTURE.....	229
FIGURE 11.39: PHYSICAL OPERATION DIAGRAM	230
FIGURE 11.40: PHYSICAL BUS MODEL	230
FIGURE 11.41: CHANNEL 'X' LAYER.....	231
FIGURE 11.42: PROPAGATION DELAY CALCULATION BLOCKS	232
FIGURE 11.43: DELAY SLOTS BLOCKS	232
FIGURE 11.44: ADDITIONAL FRAMES LAYER.....	232
FIGURE 11.45: FRAME ROUTING BLOCK	233
FIGURE 11.46: APPLICATION LAYER OPERATION	234

TABLE OF FIGURES

FIGURE 11.47: APPLICATION LAYER	234
FIGURE 11.48: DRIVER OPERATION	235
FIGURE 11.49: SOFTWARE DRIVER LAYER	235
FIGURE 11.50: SOFTWARE DRIVER DELAY	236
FIGURE 11.51: BUS MONITOR MODEL.....	237
FIGURE 11.52: BUS MONITOR MODEL.....	237
FIGURE 12.1: MODEL DEVELOPMENT FLOW CHART	240
FIGURE 12.2: MODEL SUBSYSTEM BLOCK DIAGRAM.....	244
FIGURE 12.3: SYNCHRONISATION TEST 1 ATTRIBUTE SCOPE GRAPH FOR CYCLE ENTITIES.....	246
FIGURE 13.1: CALIBRATION ITERATIVE PROCESS	262
FIGURE 13.2: SIMULATION MODEL DEVELOPMENT PROCESS.....	263
FIGURE 13.3: TOP DOWN VIEW OF THE FUJITSU SK-91F467-FLEXRAY DEVELOPMENT BOARD	265
FIGURE 13.4: FLEXTINY MODULE	266
FIGURE 13.5: PASSIVE STAR.....	267
FIGURE 13.6: VECTOR VN3600 USB INTERFACE FOR FLEXRAY	267
FIGURE 13.7: DESIGNER PRO MAIN WINDOW	270
FIGURE 13.8: THE FIRST PAGE OF THE FLEXRAY CONFIGURATION WIZARD	271
FIGURE 13.9: THE SECOND PAGE OF THE FLEXRAY CONFIGURATION WIZARD	271
FIGURE 13.10: SOFTUNE WORKBENCH MAIN WINDOW	272
FIGURE 13.11: FLEXCONFIG MAIN WINDOW	273
FIGURE 13.12: CANALYZER.FLEXRAY TRACE WINDOW.....	274
FIGURE 13.13: E-RAY STRUCTURE.....	277
FIGURE 13.14: E-RAY STATUS REGISTER INTERRUPTS	278
FIGURE 13.15: FREE-RUN TIMER SETTINGS	280
FIGURE 13.16: CALIBRATION HARDWARE SETUP	281
FIGURE 13.17: FLOW DIRECTIONS OF DATA IN A FLEXRAY SYSTEM	282
FIGURE 13.18: FLOW DIRECTIONS OF DATA IN A FLEXRAY SYSTEM	283
FIGURE 13.19: CALIBRATION HARDWARE SETUP – REVISED	288
FIGURE 13.20: FUJITSU FLEXRAY DRIVER TRANSMIT TIMINGS.....	292
FIGURE 13.21: FUJITSU FLEXRAY DRIVER TRANSMIT TIMINGS WITH LINEAR TREND LINE.....	292
FIGURE 13.22: FUJITSU FLEXRAY DRIVER TRANSMIT TIMINGS WITH POLYNOMIAL TREND LINE	293
FIGURE 13.23: FUJITSU FLEXRAY DRIVER RECEIVE TIMINGS	294
FIGURE 13.24: FUJITSU FLEXRAY DRIVER RECEIVE TIMINGS WITH LINEAR TREND LINE.....	295
FIGURE 13.25: COMMSTACK TRANSMIT TIMINGS	295
FIGURE 13.26: COMMSTACK TRANSMIT TIMINGS WITH POLYNOMIAL TREND LINE	296
FIGURE 13.27: COMMSTACK TRANSMIT TIMINGS WITH LINEAR TREND LINE	296
FIGURE 13.28: COMMSTACK RECEIVE TIMINGS	297
FIGURE 13.29: COMMSTACK RECEIVE TIMINGS WITH LINEAR TREND LINE	298

TABLE OF FIGURES

FIGURE 13.30: TRANSMIT INTERRUPT TIMING WITH LINEAR TREND LINE	298
FIGURE 13.31: TRANSMIT TIMES	299
FIGURE 13.32: TRANSIENT BUFFER TRANSFER TIMES	300
FIGURE 13.33: RECEIVE INTERRUPT TIMINGS.....	301
FIGURE 13.34: BUFFER UPDATE TIMINGS	301
FIGURE 13.35: IBF INTERRUPT TIMINGS WITH SERIES TREND LINE.....	303
FIGURE 13.36: IBF TIMINGS WITH LINEAR TREND LINE.....	303
FIGURE 13.37: OBF INTERRUPT TIMINGS	304
FIGURE 13.38: FFRD AMENDED RECEIVE TIMINGS WITH LINEAR TREND LINE	305
FIGURE 13.39: COMMSTACK AMENDED TIMINGS WITH LINEAR TREND LINE.....	306
FIGURE 14.1: MODEL BUILDING PROCESS	324
FIGURE 14.2: CALIBRATION ITERATIVE PROCESS	325
FIGURE 14.3: FINAL MODEL STEPS	345

TABLE OF TABLES

Table of Tables

TABLE 3.1: SAE AUTOMOTIVE NETWORK CLASSIFICATIONS.....	11
TABLE 3.2: EVENT-TRIGGERED VS. TIME-TRIGGERED SYSTEMS	14
TABLE 3.3: MOST25 FRAME BYTE SUMMARY	22
TABLE 3.4: MOST25 FRAME BYTE SUMMARY	23
TABLE 4.1: K AS A FUNCTION OF A LIST OF VALUES	51
TABLE 5.1: COMPONENTS OF SOFTWARE MEASUREMENT	78
TABLE 5.2: FOCUS TYPE DEFINITIONS FOR FLEXRAY.....	82
TABLE 5.3: SYSTEM ANALYSIS TECHNIQUE REQUIREMENTS SUMMARY	101
TABLE 6.1: MESSAGE BUFFER ASSIGNMENT.....	111
TABLE 6.2: ERROR MODES	117
TABLE 6.3: MESSAGE RAM SCAN.....	120
TABLE 6.4: CYCLE SET DEFINITION	130
TABLE 6.5: EXAMPLES OF CYCLE SETS.....	130
TABLE 6.6: CHANNEL FILTERING BIT CONFIGURATIONS.....	131
TABLE 7.1: SIMULATION SOFTWARE SELECTION ANALYSIS.....	164
TABLE 11.1: ENTITY ATTRIBUTES	206
TABLE 12.1: SYNCHRONISATION TEST 1 SIMULATION TIME RESULTS.....	245
TABLE 12.2: SYNCHRONISATION TEST 1 RESULTS SUMMARY.....	245
TABLE 12.3: VERIFICATION TEST CASE PARAMETERS	250
TABLE 12.4: VERIFICATION TEST CASE RANDOM NUMBER SEEDS	251
TABLE 12.5: VERIFICATION TEST CASE 1 RESULT SUMMARY	252
TABLE 12.6: VERIFICATION TEST CASE 2 RESULT SUMMARY	252
TABLE 12.7: VERIFICATION TEST CASE 3 RESULT SUMMARY	253
TABLE 12.8: VERIFICATION TEST CASE 4 RESULT SUMMARY	253
TABLE 12.9: VERIFICATION TEST CASE 5 RESULT SUMMARY	253
TABLE 12.10: VERIFICATION TEST CASE 6 RESULT SUMMARY	254
TABLE 12.11: VERIFICATION TEST CASE 7 RESULT SUMMARY	254
TABLE 12.12: VERIFICATION TEST CASE 8 RESULT SUMMARY	254
TABLE 12.13: VERIFICATION TEST CASE 9 RESULT SUMMARY	255
TABLE 12.14: VERIFICATION TEST CASE 10 RESULT SUMMARY	255

TABLE OF TABLES

TABLE 12.15: VERIFICATION TEST CASE 11 RESULT SUMMARY	255
TABLE 12.16: VERIFICATION TEST CASE 12 RESULT SUMMARY	256
TABLE 12.17: VERIFICATION TEST CASE 13 RESULT SUMMARY	256
TABLE 12.18: VERIFICATION TEST CASE 14 RESULT SUMMARY	256
TABLE 12.19: SPEED TESTS	258
TABLE 13.1: SK-91F467-FLEXRAY DEVELOPMENT BOARD INTERRUPT CONNECTIONS	279
TABLE 13.2: CALIBRATION TEST CASE PARAMETERS	286
TABLE 13.3: CALIBRATION TEST CASE RANDOM NUMBER SEEDS	287
TABLE 13.4: FFRD_APL_GET_TIME() TIME DIFFERENCES (μ S)	290
TABLE 13.5: FREE RUN TIMER DIFFERENCES (NS).....	290
TABLE 13.6: INTERRUPT LATENCY TIMES (μ S)	291
TABLE 13.7: TRANSMIT AVERAGES.....	300
TABLE 13.8: RECEIVE AVERAGES.....	302
TABLE 13.9: IBF AVERAGES	304
TABLE 13.10: OBF AVERAGES.....	305
TABLE 13.11: SYSTEM TIMING CONSTRAINTS	307
TABLE 13.12: CALIBRATION TEST CASE 1 DATA	309
TABLE 13.13: CALIBRATION TEST CASE 2 DATA	309
TABLE 13.14: CALIBRATION TEST CASE 3 DATA	310
TABLE 13.15: CALIBRATION TEST CASE 4 DATA	310
TABLE 13.16: CALIBRATION TEST CASE 5 DATA	311
TABLE 13.17: CALIBRATION TEST CASE 1 ANALYSIS.....	312
TABLE 13.18: CALIBRATION TEST CASE 2 ANALYSIS.....	313
TABLE 13.19: CALIBRATION TEST CASE 3 ANALYSIS.....	314
TABLE 13.20: CALIBRATION TEST CASE 4 ANALYSIS.....	314
TABLE 13.21: CALIBRATION TEST CASE 5 ANALYSIS.....	315
TABLE 13.22: CALIBRATION TEST RESULTS SUMMARY	316
TABLE 13.23: TRANSMIT PIPELINE TIMING.....	318
TABLE 13.24: RECEIVE PIPELINE TIMING	319
TABLE 14.1: VALIDATION TEST CASE PARAMETERS.....	327
TABLE 14.2: VALIDATION TEST CASE RANDOM NUMBER SEEDS.....	328
TABLE 14.3: VALIDATION TEST CASE 1 DATA	329
TABLE 14.4: VALIDATION TEST CASE 2 DATA	329
TABLE 14.5: VALIDATION TEST CASE 3 DATA	330
TABLE 14.6: VALIDATION TEST CASE 4 DATA	331
TABLE 14.7: VALIDATION TEST CASE 5 DATA	331
TABLE 14.8: VALIDATION TEST CASE 6 DATA	332
TABLE 14.9: CALIBRATION TEST CASE 7 DATA	333

TABLE OF TABLES

TABLE 14.10: VALIDATION TEST CASE 8 DATA	333
TABLE 14.11: VALIDATION TEST RESULTS SUMMARY	334
TABLE 14.12: TRANSMIT PIPELINE TIMING	336
TABLE 14.13: RECEIVE PIPELINE TIMING	337
TABLE 14.14: BUFFER UPDATE TIME	339
TABLE 14.15: BUFFER READ TIME	340
TABLE 14.16: TOTAL SOFTWARE DRIVER TIMES	342
FIGURE 14.3: FINAL MODEL STEPS	345
TABLE A.1: MODEL COLOUR CODING	IV
TABLE B.1: PHYSICAL BUS INPUT WORKSPACE VARIABLES	VI
TABLE B.2: NODE INPUT WORKSPACE VARIABLES	VII
TABLE B.3: COMMUNICATIONS CONTROLLER INPUT WORKSPACE VARIABLES	VII
TABLE B.4: PROTOCOL OPERATIONS CONTROL INPUT WORKSPACE VARIABLES	VIII
TABLE B.5: PHYSICAL BUS OUTPUT WORKSPACE VARIABLES	IX
TABLE B.6: NODE OUTPUT WORKSPACE VARIABLES	X
TABLE B.7: COMMUNICATIONS CONTROLLER OUTPUT WORKSPACE VARIABLES	XII
TABLE B.8: PROTOCOL OPERATIONS CONTROL OUTPUT WORKSPACE VARIABLES	XIV
TABLE B.9: ENTITY ATTRIBUTES	XV

Section I:

Thesis Overview

Chapter 1 . Thesis Overview

1.1 Problem Specification

The main aim of this project is to research the workings of a FlexRay node and to suggest a method to optimally configure that node within a network.

As FlexRay is a new network protocol there is a need to fully understand how best to configure the network. This is so the maximum use of the network with a minimum associated cost can be achieved. The optimisation should also be done as there are several other networking schemes, such as CAN, LIN and MOST. No one networking scheme is perfect for all applications, and more than one type of network may be needed to efficiently implement all the systems found within a car. Implementing any number of these at the same time could increase cost and ultimately lead to problems if the systems do not work well together.

FlexRay looks likely to become the networking scheme of choice for safety critical systems such as X-by-wire systems (Pop et. al. 2007, p51). It is therefore important to identify any problems or areas for improvement early on. This will lead to a wider range of applications being developed that could increase customer comfort and safety.

The building of a model will allow the testing of a node with various configurations in a faster and cheaper way than by experimentation on a real network. It is therefore necessary to understand how a FlexRay network operates and how a node interacts with the other nodes on a network in order to accurately obtain realistic data.

1.2 Research Questions

The main goal of this research is to develop a method to optimise a FlexRay node for efficient and reliable communication.

THESIS OVERVIEW

This research leads to a number of key questions that are to be answered. These questions are as follows:

- What aspects of the FlexRay controller configuration most affects the performance and design of distributed vehicle applications?
- What guidelines should be used to configure the protocol stack for best application performance?
- What techniques can be used to optimise local buffer usage for specific vehicle applications using a fixed global network message schedule?

1.3 Document Layout

The layout of this document is as follows:

- **Chapter 1 – Thesis Overview:** This chapter covers the problem specification, solution requirements and research questions in relation to this research.
- **Chapter 2 – Literary Review Introduction:** This chapter introduces the topics and criteria for discussion covered in the literary review section of this thesis.
- **Chapter 3 – Automotive Networks:** This chapter covers the current state of automotive networking technology.
- **Chapter 4 – FlexRay:** This chapter describes the FlexRay protocol.
- **Chapter 5 – Performance Analysis:** This chapter describes methods to carry out performance analysis. Different methods that have been used in the past are also introduced.
- **Chapter 6 – E-Ray:** This chapter covers the workings of the Bosch E-Ray communication controller.

THESIS OVERVIEW

- **Chapter 7 – Discrete Event Simulation:** This chapter describes the discrete event simulation method of modeling systems. Different simulation software packages are introduced and evaluated. MATLAB, the simulation package that was ultimately chosen is covered in detail. The selection process for the simulation software is also discussed.
- **Chapter 8 – FlexRay Software Drivers:** This chapter focuses on the different software drivers that are available to implement FlexRay systems.
- **Chapter 9 - Literary Review Summary:** This chapter summaries the literary review and the available literature. It also discusses the need for further research in the area of automotive networks.
- **Chapter 10 - Methodology:** This chapter covers the methodology used to carry out the research.
- **Chapter 11 - Simulation Model Development:** This chapter documents the specification and implementation of FlexRay node simulation model.
- **Chapter 12 - Verification:** This chapter discusses the steps used to verify the model.
- **Chapter 13 - Calibration:** This chapter covers the calibration procedure for the model. Test cases are outlined and the calibration process is reviewed. The equipment that was used is outlined and test results are summarised.
- **Chapter 14 - Validation:** This chapter covers the steps used to validate the simulation model. Test cases are outlined and the validation process is reviewed.
- **Chapter 16 - Conclusion:** This chapter summarises the work done during the research, conclusions drawn from the results and suggestions for areas of further study are put forward.

1.4 References

Pop, T., Pop, P, Eles, P. and Peng, Z. (2007) Bus Access Optimisation for FlexRay-Based Distributed Embedded Systems, Proceedings of the Conference on Design, Automation and Test in Europe, Nice, France, April 16-20 2007, IEEE Computer Society Washington, DC, 51 – 56.

Section II:
Literary Review

Chapter 2 . Literary Review

Introduction

2.1 Scope

The literary review introduces key concepts and topics that were looked at to successfully complete the project. The information provided allows the reader to gain an understanding of why the research is necessary. It also allows the reader to form opinions on the methods that were chosen to complete the research. The background information provided also allows the reader to understand the significance of the research.

A number of topics are covered in this literary review. The main topics covered can be summarised as:

- The main aspect of the project involves improving the performance of a FlexRay node, therefore FlexRay and its alternatives are explored.
- The Bosch E-Ray chip is the FlexRay communications controller that was available for study to this project. Its key features and implementation are described.
- The method of adapting and running tests of a FlexRay node is simulation. The reasons for this methodology to be chosen along with simulation theory are covered.
- The current state of automotive networking and the need for research in this area are introduced.
- Analysis techniques to quantify the performance of the model are discussed.

2.2 Terminology

This section outlines terminology that will be used in the following chapters. It is an alphabetical listing with brief definitions for each phrase. It should be noted that the definitions may not cover all terms that the reader is unfamiliar with. An attempt by the author has been made to give a brief explanation, within the scope of this thesis, of all the technical terms used that the reader may not be familiar with. If an explanation of a term is given elsewhere in this thesis it has been omitted from this section.

Actuator: A device that converts electrical signals into physical actions. An example of this is a D.C. motor which converts an electrical signal into a turning motion.

Application: A piece of software that defines how information is handled or processed by a computer system.

Bus: The physical wire or wires over which information is sent between two different nodes on a network (see the definition of a node given below).

Channel: This is a path through which information 'flows'. A FlexRay bus is an example of a channel.

Communications Controller: A computer chip specifically designed to transmit and receive data over a communications channel. For example, in the case of a FlexRay communications channel, data is handled according to the FlexRay protocol specifications.

Host: A microprocessor unit (MPU) that has a communications controller embedded or attached. An application on the host may send and/or receive data to/from the FlexRay bus. It may also process information in order to implement a task or function.

Multiplexing: This is a where one or more device share a common communication channel. It splits either the time or frequency spectrum available to allow the devices access to the channel.

Node: A piece of hardware that can consist of a communications controller and host MPU. There may also be attached a sensor and/or actuator. The host is attached to the communications channel via a communications controller. The application running on the node defines its function.

Register: A dedicated area of an electronic chip that stores values used to determine the working of the device or program.

LITERARY REVIEW

Sensor: A sensor reads in information from a physical device and converts it into an electrical value. An example of this is a thermistor that converts temperature into an analogue or digital signal that can be displayed on a dash board display.

X-by-wire: A method of replacing physical mechanical links with computer-assisted actuators.

2.3 Criteria for Discussion

Each chapter was included under one of the following criteria:

1. It provides necessary information to understand the need of the project.
2. It provides necessary information to understand the methods used to carry out the project.
3. It gives an understanding of the equipment and methodologies available to successfully carry out the project.

2.4 Limits of the Review

The literary review covers many topics. However there are a number of areas related to FlexRay that have not been covered. An attempt has been made to only include the necessary information to allow the reader to understand the need for the research.

Chapter 3 . Automotive Networks

3.1 Introduction

There are a number of different communication protocols that have been developed for use by the automotive industry. Each networking scheme is intended for a different purpose. This chapter introduces various networking protocols used in the automotive industry. It also attempts to outline the challenges faced by automotive networks and highlight any weaknesses in relation to the available networks.

3.2 Automotive Networks

Since the first electronic device was installed in an automotive vehicle the number of components has increased dramatically. It is estimated that up to 90% of innovations in the automotive industry are due to electric and electronic systems (Fennel 2006). This is set to increase further with new applications such as x-by-wire applications.

Figure 3.1 (TechInsights 2008) shows how the increase in the number computer components leads to an exponential increase of the number of connections needed to connect each device. Without employing a serial communications network to connect each ECU the increased number of wires would become impractical. Each of the communications networks described in this chapter uses serial transmission over a small number of wires. This allows all the nodes on the network to be connected while reducing the number of individual point-to-point connections.

LITERARY REVIEW

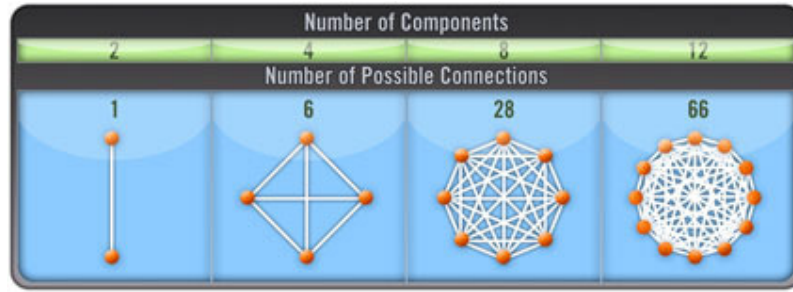


Figure 3.1: Computer components and possible connections

Navet et. al. (2005) describes how in 1994 the Society for Automotive Engineers (SAE) defined a classification for automotive networks. Every automotive networking protocol belongs to one of the SAE classes of automotive networks. Table 3.1 details the classifications.

Class	Functions	Bit Rate	Example Protocols
A	Simple, low-cost, control applications.	<10kb/s	LIN, TTP/A
B	Inter-ECU communication applications	10 – 125 kb/s	J1850, low-speed CAN
C	Powertrain/chassis applications	125 kb/s – 1 Mb/s	High-speed CAN
D*	Multimedia applications, X-by-wire, Fault tolerant applications	> 1 Mb/s	MOST, TTP/C, FlexRay

Table 3.1: SAE automotive network classifications

Usually the higher the classification of a particular networking protocol, the more complex it becomes. This complexity comes with advantages and drawbacks. For instance FlexRay is more complex than LIN; however FlexRay provides a higher bit rate and the ability to transmit data in both a time-triggered and event-triggered manner. This will increase the cost in terms of setup time and the actual cost of components while providing greater data throughput.

* Class D is not formally defined. However it is considered to be networks operating over 1Mb/s (Navet et. al. 2005).

3.3 Networking Type Overview

Each of the protocols described in this chapter can be classified as either an event-triggered or time-triggered system. A time-triggered network sends messages at fixed points in time. Event-triggered systems send messages in reaction to stimuli. For instance if a person wishes to open a window in a car they might press a button. This event will then generate a message to operate a motor to control the window. This section highlights some problems and benefits of both types of system.

Event-triggered: Event-triggered messages have unpredictable transmission patterns; this makes analysis of performance relatively difficult. However for sporadic transmission behavior this is a good implementation and leads to a flexible system (Kopetz 2000). A comparative study of time-triggered and event-triggered systems found that, during heavy bus loading, event-triggered messages may fail to transmit due to higher priority messages blocking lower priority messages. However when an average delay is taken of the messages sent, the event-triggered protocols experienced a shorter delay (Claesson et. al. 2003). This may be due to the fact that higher priority messages may not occur as frequently as lower priority events. From the point of view of resource utilisation this leads to event-triggered systems being superior but they do not scale as easily as time-triggered systems, this is due to a lack of any ‘temporal firewall’ (Kopetz 1991). A temporal firewall is a way to prevent unwanted communications between the different nodes on a network by the multiplexing of time slices to allow or deny communication.

Time-triggered: Time-triggered messages have predictable transmission patterns; this makes for easier performance analysis. Interoperability of the different nodes in the network is also an advantage achieved from employing this method as each node is given a specific time slot to transmit. Time-triggered systems are also ideal for real-time systems where deadlines must be met (Kopetz 2000). The design stage of a time-triggered system can be more complicated compared to a similar system implemented as an event-triggered system. This is because timing constraints must be met to ensure information is sent out before a deadline. This leads to a more detailed planning phase where timing constraints of all aspects of a system should be considered. It is necessary that an application running on a particular node transmits any data to any interested nodes within a given time. This design stage will however lead to a reduced verification time of the time-triggered system (Kopetz 1991).

Event-triggered vs. Time-triggered: Scheler and Schröder-Preikschat (2006) compare event-triggered and time-triggered architecture. They looked at analysability, predictability, testability, extensibility, fault-tolerance and resource utilisation. A summary of this breakdown can be seen in Table 3.2. It can be concluded from their findings that neither approach is sufficient for every system. If the data is sporadic then event-triggered protocols will be a good approach under low bus loadings. However if the system is a real-time system and must adhere to strict timing constraints, then time-triggered protocols should be used. However the development process may be longer in this case.

Figure 3.2 shows the traffic patterns for event-triggered and time-triggered systems. In event-triggered system messages may attempt to gain access to the communications bus at the same time. The message with the highest priority will gain access to the bus. Other messages must then wait until the communications bus is free before again attempting to gain access to the bus. In a time-triggered system a message is assigned to a slot at the design time. In Figure 3.2 Message 1 has the highest priority and message 3 the lowest Priority. The time-triggered messages are all represented by different colours. The slots may make use of a multiplexing technique to allow different messages to be transmitted during the same time slot but over different communications cycles. It can be seen that the same message is transmitted during the same time slot every communication cycle if multiplexing of the slots is not implemented. Multiplexing of slots must be set at design time also.

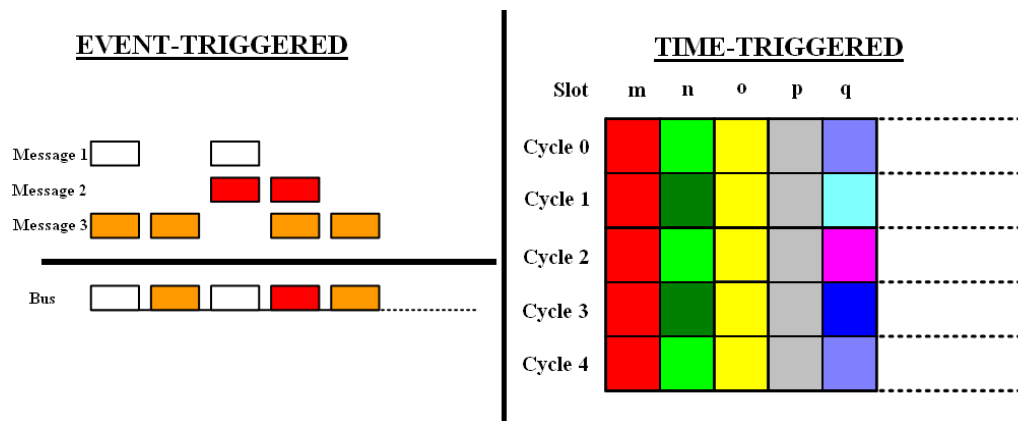


Figure 3.2: Event-triggered and time-triggered network patterns

LITERARY REVIEW

	Time-Triggered	Event-Triggered	Conclusion
Analysability	Statistically computed schedules are used to analyse the schedulability.	Response time analysis technique is used need to analyse the schedulability.	Neither method provides a better solution as detailed knowledge is necessary to perform the analysis.
Predictability	Easily analysed for predictability.	Dynamic response to events makes the system less predictable. A system may still be deterministic.	Time-triggered systems make analysis of communication patterns easier as this is set at design time.
Testability	Best to test for worst case performance. Typical load scenarios are not sufficient to test properly.	Best to test for worst case performance. Typical load scenarios are not sufficient to test properly.	Neither method is easier to test.
Extensibility	The need to recalculate static schedules is necessary if adding functions.	The response time analysis will need to be recomputed if added functionality is introduced to the system.	Neither method makes it easier to extend the functionality of the system.
Fault-Tolerance	Different nodes can make the same decision at the same time.	It is harder to achieve a fault-tolerant system unless a leader-follower system is used.	In general time-triggered systems provide more fault-tolerance within a system.
Resource Utilisation	A node is seen as redundant during any communication cycle if it has nothing to send. This means there is wasted resources.	A resource will only be requested when needed.	Event-triggered systems can maximise the resource in more cases than time-triggered systems

Table 3.2: Event-triggered vs. time-triggered systems

3.4 Automotive Network Protocols

Figure 3.3 (Leen and Heffernan 2002, p89) shows a breakdown of the types of systems implemented electronically in cars. Figure 3.4 (Denner et. al. 2004) shows the functional area breakdown for each networking scheme.

LITERARY REVIEW

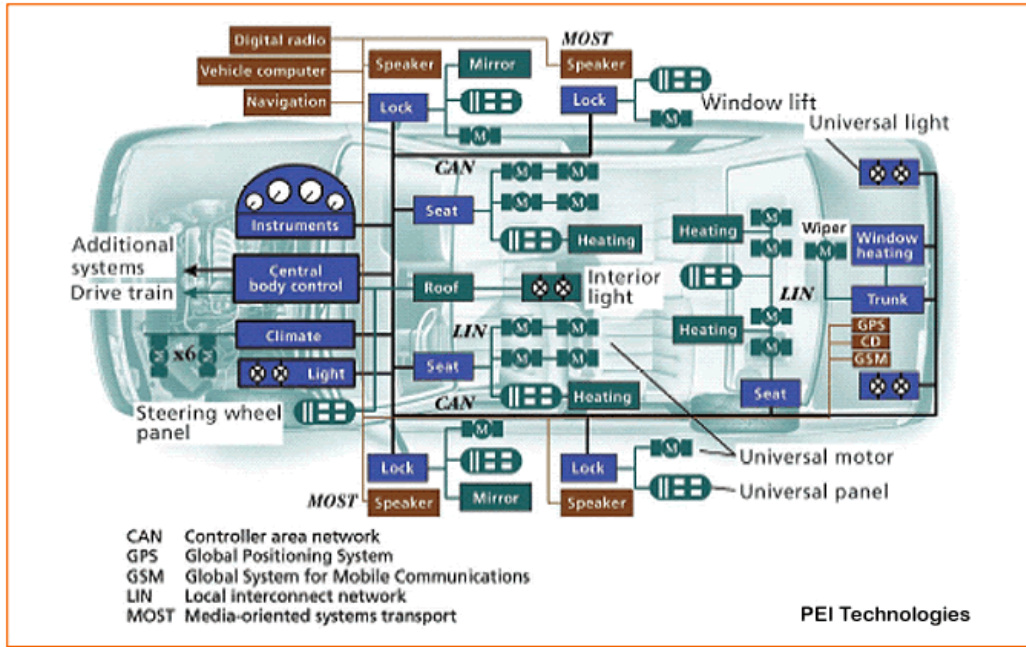


Figure 3.3: Automotive network applications

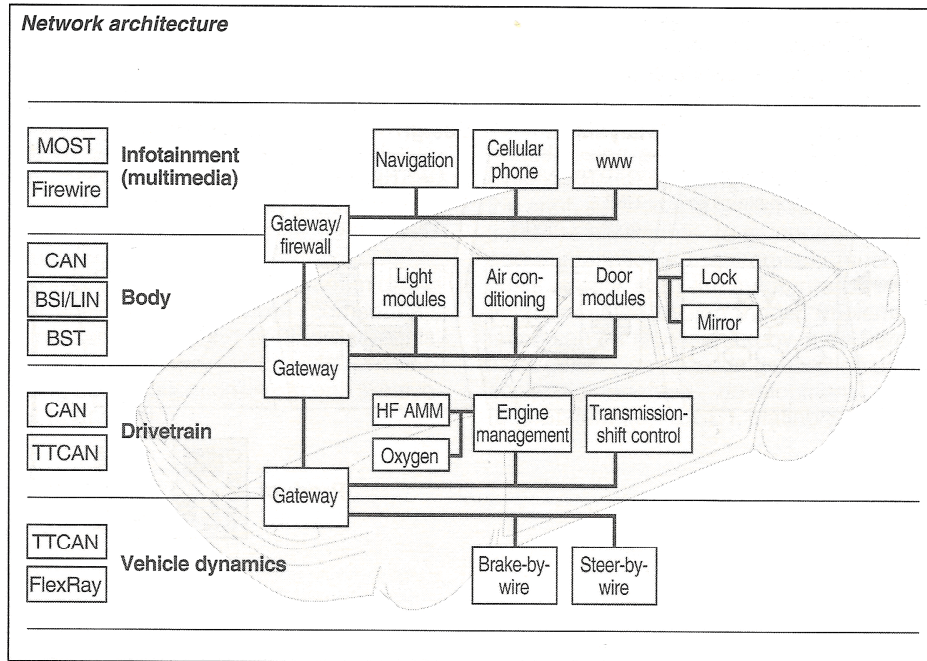


Figure 3.4: Automotive networks functionality breakdown

As can be seen from Figures 3.3 and 3.4, there is no one networking scheme that is designed to carry out all necessary communications. It is can also be seen that with

LITERARY REVIEW

the present situation, automotive electronic and electrical systems need a proper communications network to communicate. Without any multiplexed communications systems there would be a huge amount of wiring dedicated to the transmission of information between two specific nodes. The inclusion of an interconnected multiplexed communication network also reduces the number of duplicate sensors in a vehicle. In a multiplexed networking scheme sensor data can be shared to a number of different nodes all at the same time. The reduction in the number of duplicate sensors has a cost saving benefit for the manufacturer and customer. Without a multiplexed networking system a vehicle would be seen as having a drastic weight increase and thus relatively poor performance of the vehicle, in terms of both power and fuel economy, when compared to a vehicle where a multiplexed networking system is implemented.

3.5 Event-Triggered Protocols

3.5.1 Controller Area Network (CAN)

The CAN networking scheme was first introduced in 1986. It was developed by Bosch with help from Mercedes-Benz and Intel. The development of the protocol was started as early as 1983 in a bid to increase functionality for the automotive industry. The reduction in wiring within a vehicle was a consequence of the protocol. Since it was introduced it has been used in a wide range of applications within cars as well as in other areas. Most cars produced in Europe will contain at least one CAN. It has been used in trains, ships and industrial control applications (CiA 2007). It has even been implemented in the 2008 BMW RG 1200 GS Adventure motorcycle (BMW Motorrad USA 2008).

In 1991 the CAN specification 2.0 was published by Bosch. In 1993 CAN was standardised as ISO 11898 by the International Organisation of Standards with an extended frame format being standardised with an amendment in 1995. A time triggered communication protocol for CAN (TTCAN) was developed in 2000 (CiA 2007).

3.5.1.1 CAN Protocol

This section is a combination of information from Denner et. al. (2004), Carley (2006), CiA (2006), Schofield (2006), Robert Bosch GmbH (1991) and Jurgen (1999).

The CAN bus is made up of a number of ECUs that all have a priority rating. A CAN bus can be seen in Figure 3.5 (Ecartec Ltd. 2008). Figure 3.5 depicts different nodes connected onto the same CAN bus. Each node performs a different function.

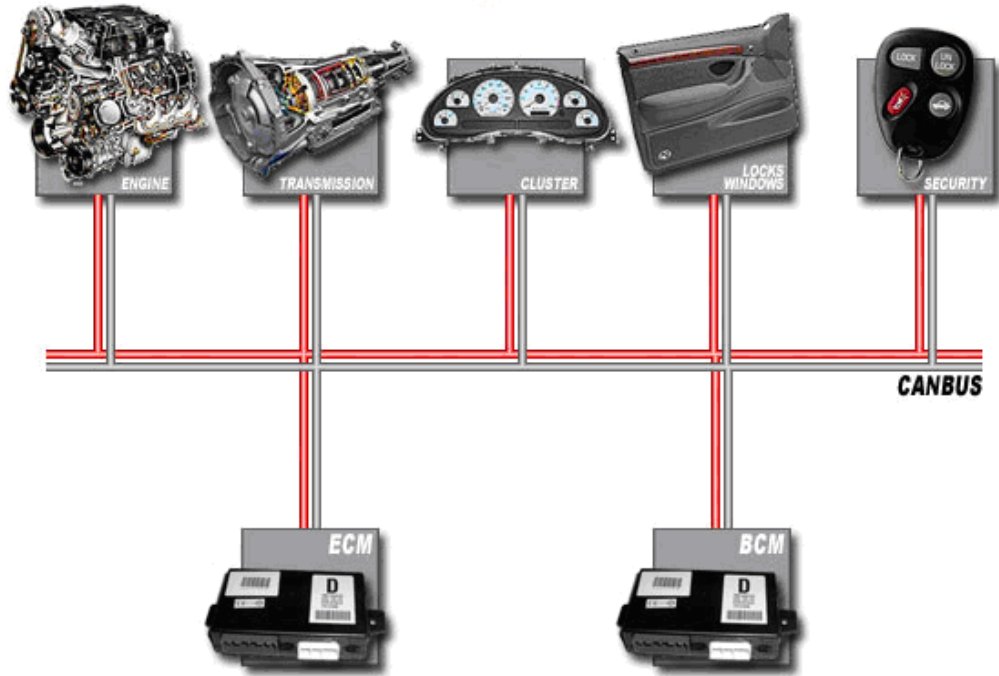


Figure 3.5: CAN Bus

To determine which node may communicate at any one time an identifier field is used in the message frame. The node with the highest priority will be allowed to transmit its message. When several nodes attempt to transmit a message at the same time the message with the highest priority will gain access without any delay. This is due to the ‘wired-AND’ bus arbitration. As the arbitration is based on a logical ‘AND’ operation, the lowest the message identifier has the highest message priority. In this way dynamic transmission is achieved. The diagram below, Figure 3.6 (Softing 2008), shows how a logic ‘0’ ensures that a low message identifier ensures a higher priority.

LITERARY REVIEW

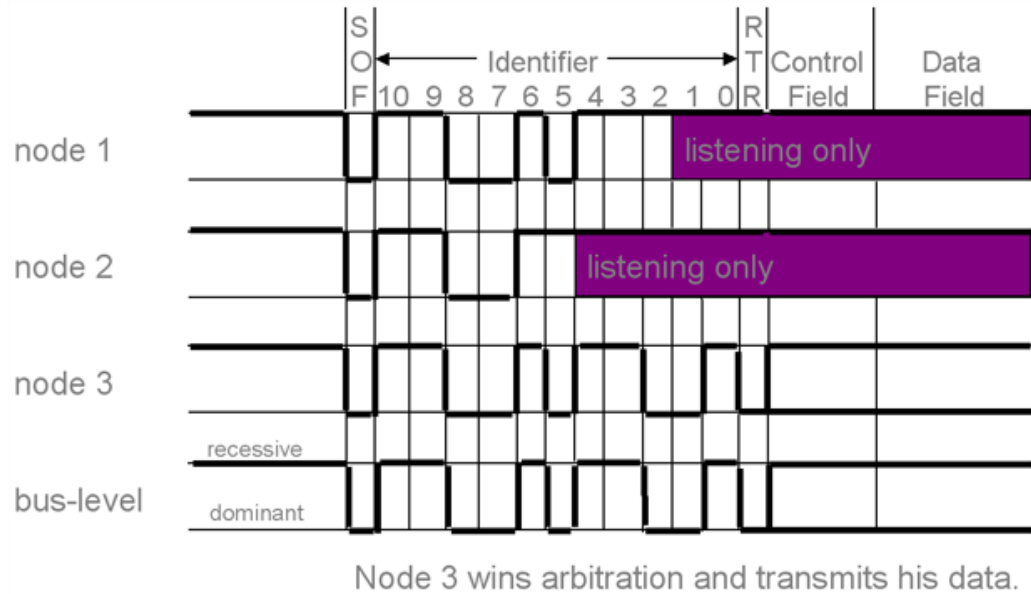


Figure 3.6: CAN bus arbitration

After a message is transmitted the nodes can again attempt to gain access to the bus.

The CAN protocol supports two frame formats, standard and extended. The message format is similar for both protocols, differing only in the number of identifier bits. Figure 3.7 (Schofield 2006), shows the standard frame format, which contains 11 identifier bits. For the extended frame format the main difference is that the identifier contains 29 bits. This means that the frames vary in length between 130 to 150 bits (maximum). The data segment however is limited to 0 to 64 bits (8 bytes).

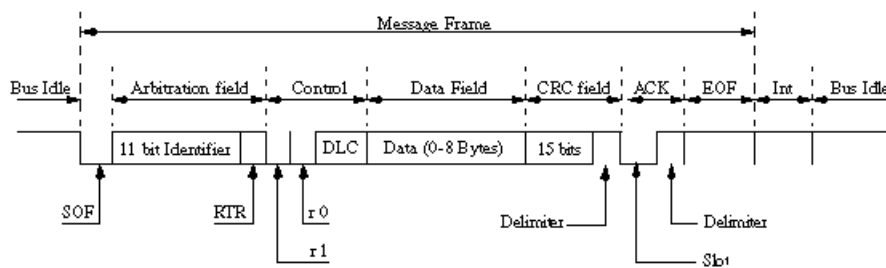


Figure 3.7: CAN standard frame format

The segments of a CAN message are as follows:

- SOF: Start of Frame bit.
- Arbitration field that consists of the identifier and a remote frame indicator.

LITERARY REVIEW

- The control field indicates the number of data bytes in the message.
- The data field contains 0 – 8 bytes of data.
- The CRC field is a 15 bit cyclic redundancy check (CRC) used by the receiving node to detect any errors in the received frame.
- The ACK field is to allow all receivers to acknowledge error free reception of the message.
- The end of frame bits indicate the end of transmission of the frame.
- Int is the inter frame space where data is not to be transmitted to ensure frame integrity.

3.5.2 Local Interconnect Network (LIN)

The local interconnect network (LIN) is a deterministic system for ECU communication with sensors, actuators and controls. The LIN specification version 2.1 was released in 2006. In August of 2004 the Society of Automotive Engineers (SAE) released J2606 which recommends a practice for implementing LIN (Vector Informatik GmbH 2008, p1).

A LIN network always consists of one master node and a number of slave nodes. It is designed so it can easily be interfaced, through a gateway, to other communication busses such as CAN (Ahlmark 2000, p1). Figure 3.8 (Ahlmark 2000, p4) shows a LIN bus configuration.

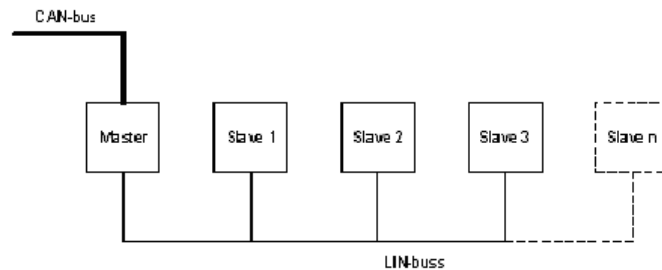


Figure 3.8: LIN bus with single master node and 'n' slave nodes

The LIN protocol operates as a Master/Slave configuration. Only the master is able to initiate communication. A LIN frame consists of a header and response sections. Communication with a slave involves the master sending the header part of a message. If the master wants to send data to the slave it continues to send the response part. If the

LITERARY REVIEW

master requests data from the slave the slave sends the response part (STMicroelectronics 2001, p4).

The header section of the frame consists of a break field, sync field and a frame identifier. The frame identifier uniquely defines the frame. The break field is used to identify the start of a transmission and the sync field is to allow receiving nodes to synchronise with the transmitted bits (LIN Consortium 2006, p29). The slave task, configured to provide the response associated with the frame identifier, will begin transmission as depicted in Figure 3.9 (LIN Consortium 2006, p13). The response consists of a data field and a checksum field. All slave nodes interested in the data associated with the frame identifier receives the response, verifies the checksum and uses the data received (LIN Consortium 2006, p13). This broadcast scheme operates at speeds up to 20kbits/s (Ahlmark 2000, p4).

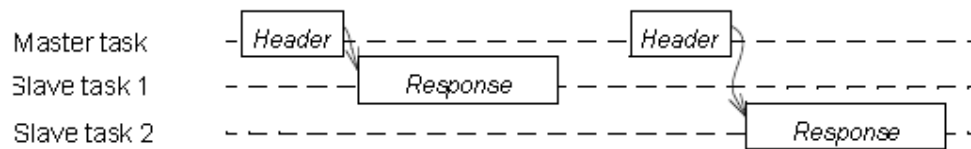


Figure 3.9: LIN communication

3.5.3 Media Oriented Systems Transport (MOST)

Media Oriented Systems Transport (MOST) is a protocol that has been developed to handle high volume data transfer. This is usually in the form of ‘infotainment’ data for audio and visual devices. Currently MOST is implemented using a plastic optical fiber (POF) communication bus. This provides a number of advantages such as weight saving and protection from electromagnetic interference. However there is also an electrical physical specification (TechInsights 2008).

MOST does not require a reconfiguration of the topology if new systems are added. The extra components can simply be added into the network by adding a connector to the physical bus. In this way MOST can be seen to have a plug and play approach to system integration (TechInsights 2008).

MOST can operate at two different speeds. There are 25Mbit/s and 50Mbit/s speed grades specified (MOST Cooperation 2006, p16). There are also a number of

LITERARY REVIEW

different communication channels over which data can be transmitted. These are as follows (MOST Cooperation 2006, pp17-18):

- Control Channel: This channel is used for small data frames with ‘bursty’ like transmission. The data rate for this channel is a relatively low 10kbits/s. The data is transported to a specific address and is protected by a CRC just like the packet data channel.
- Streaming Data Channel: This is used for continuous data such as data from an audio or video device.
- Packet Data Channel: This channel is defined for large ‘bursty’ traffic. This could be in the form of navigational map images.
- Management Streaming/Package Bandwidth: In a MOST system the management streaming and packet data streaming can be allocated space on the overall bandwidth.

A MOST system can have up to 64 nodes. Any of these nodes can be the TimingMaster and all the other nodes are Slaves. The TimingMaster provides generation and transportation of the system clock, the frames, and blocks. All Slave devices derive their clock from the MOST bus (MOST Cooperation 2006, p106). In this way MOST is a Master/Slave protocol.

One MOST25 (the 25Mbit/s variant) frame consists of 64 bytes. The first byte is used for administrative purposes. The next 60 bytes are used for Stream and Packet Data Transfer. A Boundary section of the header defines in 4 byte steps the number of data bytes. The Boundary value can only have values between 6 and 15. This means at least 24 bytes are available for Stream data transfer. All Stream data bytes are transmitted before the Packet data bytes. The next two bytes of each frame are reserved for Control data and the last byte is another administrative byte (MOST Cooperation 2006, p108). Figure 3.10 (MOST Cooperation 2006, p108) and Table 3.3 (MOST Cooperation 2006, p108) show and describe the MOST25 data frame.

LITERARY REVIEW

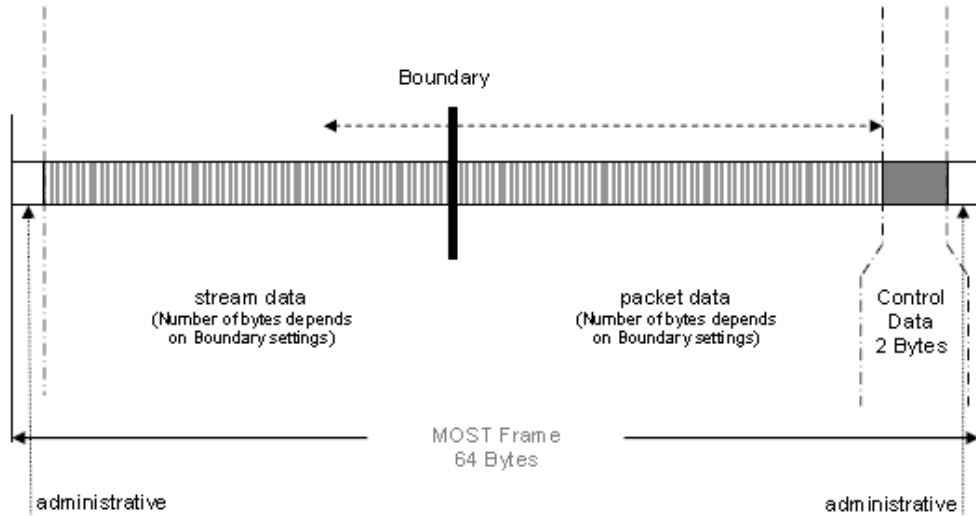


Figure 3.10: MOST25 frame

Byte Number	Task
0	Administrative <ul style="list-style-type: none"> * Preamble (bits 0-3) * 4 bits Boundary Descriptor (bits 4-7)
1 - 60	60 data bytes
61 - 62	2 data bytes for Control Messages
63	Administrative <ul style="list-style-type: none"> * Frame control and status bits * Parity bit (last bit)

Table 3.3: MOST25 frame byte summary

MOST50 is designed for high bandwidth, and one MOST50 frame consists of 128 bytes. The first 11 bytes are used for administrative purposes. Within this 4 bytes are used for Control data. The Control Message length can vary depending on the actual control message to be sent. Better utilisation of the bandwidth regarding Control Messages is obtained in this way. The next 117 bytes are used for Packet and Stream data transfer (MOST Cooperation 2006, p108). Figure 3.11 (MOST Cooperation 2006, p109) and Table 3.4 (MOST Cooperation 2006, p109) show and describe the MOST50 data frame.

LITERARY REVIEW

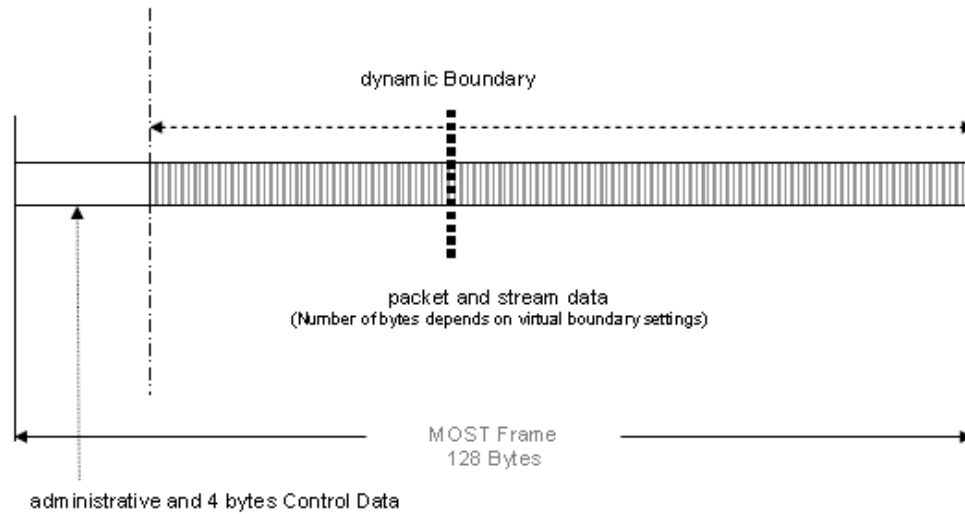


Figure 3.11: MOST25 frame

Byte Number	Task
0 - 10	Administrative, includes 4 Control data bytes, additionally ◊ System Lock flag ◊ Boundary Descriptor
11 - 127	117 data bytes

Table 3.4: MOST25 frame byte summary

3.6 Time-Triggered Protocols

Car manufacturers faced a problem when attempting to implement new safety-critical applications in cars using event-triggered communication systems. For instance, in a brake-by-wire system it is important to determine the greatest latency experienced in the system. It is important to know that when a driver presses the brake pedal the system will respond within a given time. The most widely used communication protocol in the automotive industry (CAN) is an event-driven communication protocol. This leads to an inability to determine the worst case scenario. Higher priority messages could potentially always block a message. This led to the need for a new protocol specification. By dividing up the available bandwidth into time slots a more deterministic protocol could be achieved. Two time-triggered protocols will be discussed in this section, namely TTP and FlexRay.

3.6.1 Time-Triggered Protocol (TTP)

This section is based on the works of Böhm (2005), Elmenreich and Ipp (2003) and Elemenrich and Krywult (2005).

TTP/A and TTP/C are two real-time protocols based on a TDMA scheme. TDMA (Time Division Multiple Access) is a method of multiplexing a single communication medium. To allow multiple nodes in a network to gain access to the communications bus each node is allocated a time slice. During this time slice or slot the node may transmit its message. If a node wants to transmit a message it must wait for the assigned slot to come around. When a node is not transmitting a message it can receive messages from other nodes. TTP/C is intended for connecting a number of nodes to achieve a dependable real-time system. TTP/A is a lower cost version and has reduced functionality. It is intended to use TTP/A as a bus to connect sensors and actuators.

In TTP/C the frame size can vary between 2 and 240 bytes. Each frame can carry a number of messages. In different communication rounds different messages can be transmitted during a node's allocated slot. The data is protected by a 24 bit CRC. To ensure each node sends its frame during the correct time slot the use of a bus guardian is employed. This is a separate component to the communication controller. The bus topology can be seen in Figure 3.12 (Elemenreich and Ipp 2003,P2). A star topology can also be employed. The star topology implements two stars that also act as central guardians for the network. In Figure 3.12, the CNI layer is a connection between the communications controller and the host computer. This encompasses all necessary physical connections as well as any software driver used.

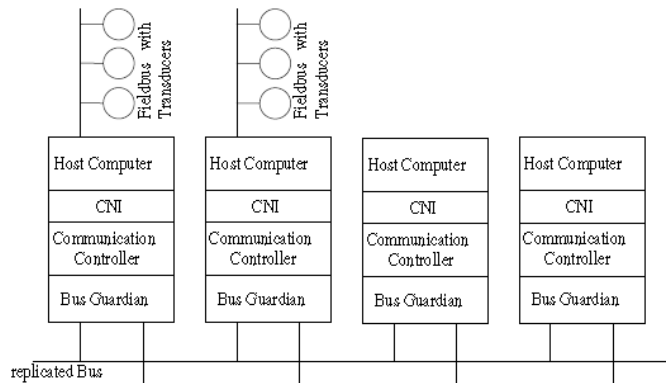


Figure 3.12: TTP bus topology

LITERARY REVIEW

To ensure that each node transmits at the allocated time the bus guardian must have a global view of the time. The current view of the global time is obtained using a clock synchronisation algorithm. This algorithm determines the current time based on the arrival of frames from other nodes and the expected time of arrival of the frames. Figure 3.13 below shows the TTP communication cycle (Elemenreich and Ipp 2003, P4).

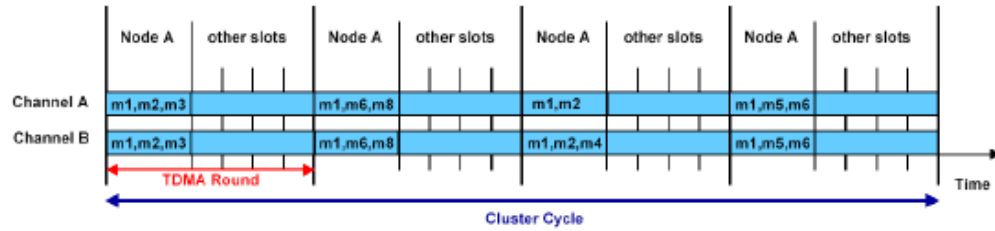


Figure 3.13: TTP communication cycle

From the TTA Group website (TTA-Group 2008), it can be seen that TTP is still found in various applications. The main area that TTP is used for appears to be in aerospace applications.

3.6.2 FlexRay

The FlexRay protocol was developed after BMW and DaimlerChrysler worked together to develop a networking scheme for future developments such as drive-by wire applications. The partnership soon led to a protocol specification which is the basis for FlexRay systems (FlexRay Consortium 2007). Due to the fact that BMW was heavily involved in the development and its similar characteristics, FlexRay can be seen as a legacy protocol of 'byteflight'. Byteflight was also developed by BMW and uses both time-triggered and event-triggered access to the communication bus (BMW 2000).

Like TTP, FlexRay is based on a TDMA approach. If a node wishes to transmit a message it must wait until its communication slot comes around. It may then transmit a single message that can consist of a data section of between 0 and 254 bytes. The header of a message is protected by an 11 bit CRC while the frame as a whole is protected by a 24 bit CRC. The frame format can be seen in Figure 3.14 (FlexRay Consortium 2005, p.90). The communication cycle is divided into 4 segments. These

LITERARY REVIEW

segments allow the nodes to transmit in a time-triggered way as well as an event-triggered fashion.

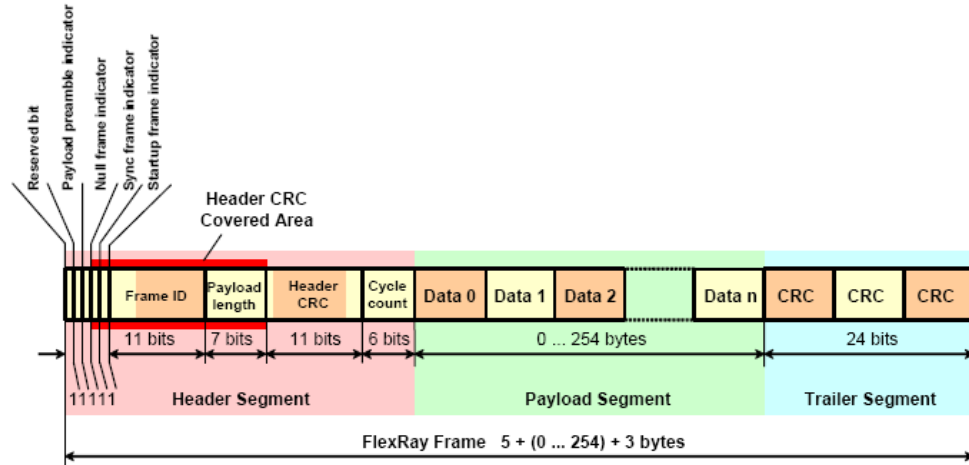


Figure 3.14: FlexRay frame

A FlexRay network supports bus and star topologies just as TTP does. However it also supports hybrid topologies. This can be seen in Figure 3.15 (FlexRay Consortium 2005, p.24).

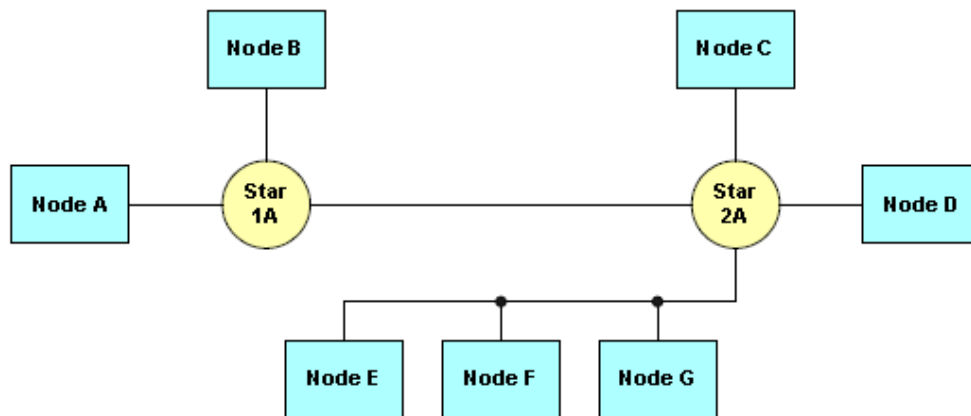


Figure 3.15: FlexRay hybrid topology

3.6.2.1 FlexRay Characteristics

The FlexRay protocol differs from TTP in its communications cycle structure. The characteristics of FlexRay will help to distinguish itself from other networking

LITERARY REVIEW

schemes and these are briefly introduced in this section. The FlexRay protocol is discussed in more detail in chapter 4.

Figure 3.16 shows an example of a bus access scheme for FlexRay (Vector Informatik GmbH 2006). The first thing that should be noted is the use of two channels. A node may be allowed to transmit a frame of data on one or both of these channels. FlexRay also allows for a node to transmit a frame on a particular channel while a different node transmits data during the same slot but on the other channel. This must be agreed before implementation during the design stage. This configurability creates an efficient use of the available bandwidth. As a node may transmit the same data on both channels during its allocated slot FlexRay also provides the ability to be configured with redundancy as standard.

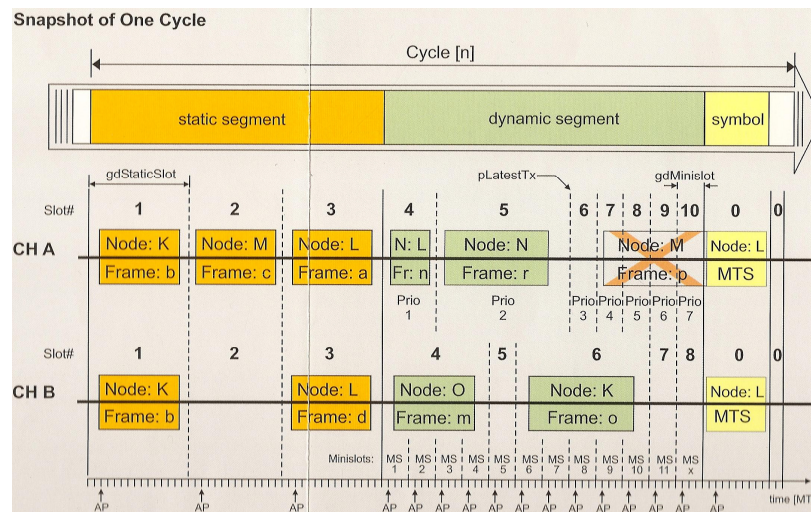


Figure 3.16: FlexRay bus access

The second piece of information that can be drawn from Figure 3.16 is that the communications cycle is split into a number of sections. In the diagram these are named as the static segment, the dynamic segment and symbol. There is a fourth segment not named in the diagram. This segment is the network idle time.

The static segment consists of slots that are of fixed length. The dynamic segment allows a frame to be transmitted with a variable length. To allow this a mechanism was developed. The dynamic segment uses minislots to form a type of flexible TDMA scheme (FTDMA) (Heller et. al. 2008, p206). Minislots are smaller in size than static slots. A dynamic frame can however transmit over a number of

LITERARY REVIEW

minislots. The slot counter will not increment until the frame has finished transmitting its data. A node that wishes to transmit a frame must still wait for its slot to transmit in the dynamic segment. If a node has no frame to transmit during its assigned dynamic slot then a single minislot time will expire before the slot counter is incremented. A node must also ensure that there is sufficient time to transmit its frame before the end of the dynamic segment. If there is insufficient time to do so, the node must wait until the next communication cycle before attempting to do so again.

Another aspect of the FlexRay communication protocol is the ability to assign different frames to the same slot. This is done by using the same slot but during a different communication cycle. There are 64 cycles and they are numbered and range between 0 and 63. When the 64th cycle completes the cycle count is reset to 0 and the whole process starts again. This can be seen in Figure 3.17.

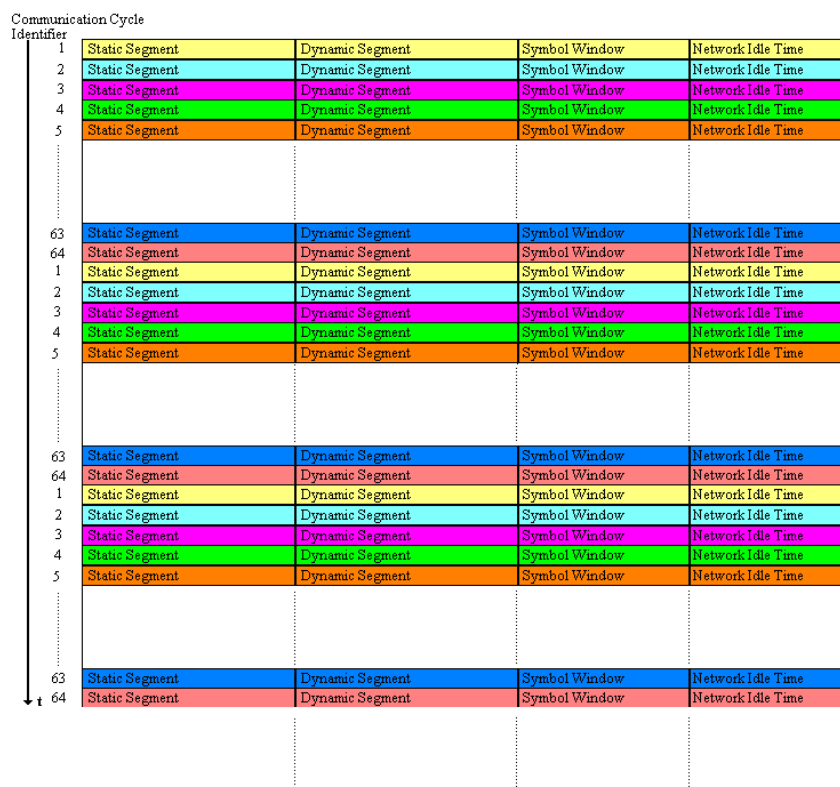


Figure 3.17: Cycle multiplexing

In FlexRay there is a need to know the current communication time. This is necessary to avoid any potential conflicts when nodes attempt to transmit their frames. Unlike other networking schemes where the current time is transmitted by a single node,

LITERARY REVIEW

every FlexRay node keeps track of the current time. Figures 3.18-3.20 (inclusive) (Vector Informatik GmbH 2006), convey the mechanism that is used to synchronise all nodes in a cluster. A number of nodes are setup to transmit 'sync' frames. These are then used by all nodes in a cluster to obtain the timing information.

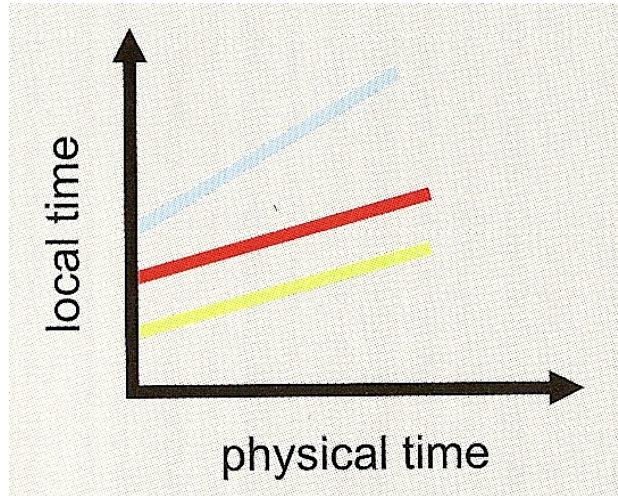


Figure 3.18: Rate differences

The sync algorithm checks the rate at which the nodes local clock is advancing. The arrival times of the sync frames are then compared to this. If the nodes clock is running faster or more slowly than it should be, measures are taken to correct this. Figure 3.18 illustrates how three nodes' global time advance rate could be different. This is known as rate correction in FlexRay.

The offset to the node's view of arriving frames to the global time is also checked. If it is found that the node views slots as beginning before or after the arrival of the sync frames, then again a corrective action can be taken. Figure 3.19 shows the offset differences of three nodes. The method of correcting this is known as offset correction in FlexRay.

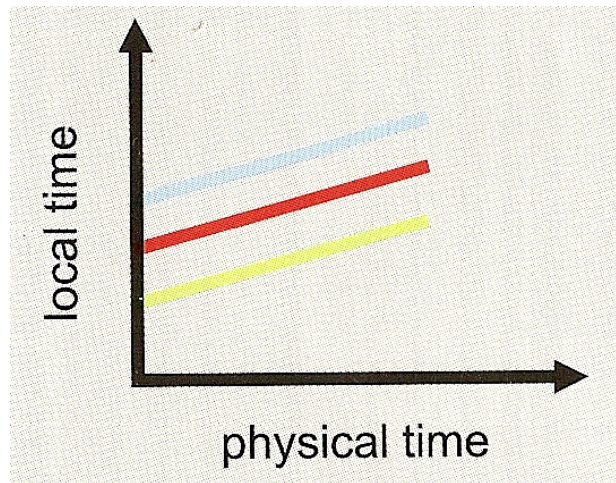


Figure 3.19: Offset differences

Figure 3.20 shows that when both rate and offset corrections are applied, then all nodes should share a common view of the global time. These checks must be conducted while any communication is taking place. If this does not happen then differences in local oscillator components will cause a divergence in the view of the global time.

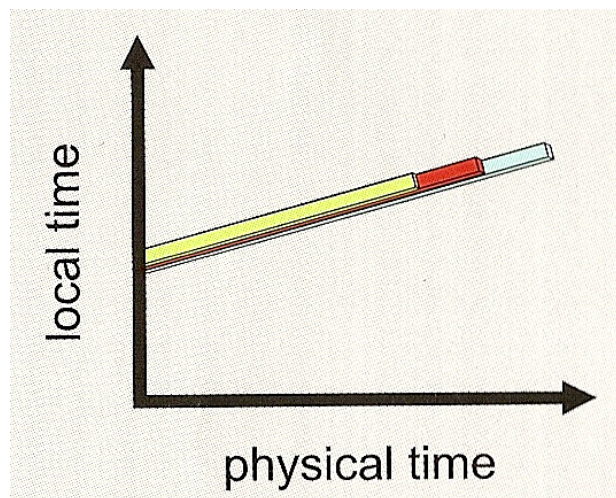


Figure 3.20: Rate and offset correction applied

3.7 Automotive Network Design

There are a number of different networking protocols and as such a network designer must choose the most appropriate protocol(s) to meet their needs. This is the

LITERARY REVIEW

first step to creating a reliable and efficient networking system. However once the networking scheme has been chosen there is still a need to define the systems configuration and constraints. This involves for example assigning messages to frame IDs. In the standard CAN frame format there are an 11-bit identifier field. This allows 2,048 different messages to be assigned to the various nodes implemented in a vehicle. The extended version specifies a 29-bit identifier which produces 536,870,912 possible messages. FlexRay also uses an 11-bit identifier field for frames. If all of these identifier IDs are used there will be a large number of messages to assign to different nodes and systems. All communications protocols require careful configuration to achieve a suitable and efficient networking system. If the configuration is not done correctly then errors may be observed in the system.

A number of tools and methodologies to configure or monitor CAN systems already exist. This is due to the age and knowledge of the protocol. FlexRay is a newer communications protocol with a limited number of implementations. To assist developers to optimise a FlexRay based system there have been a number of tools developed to ease the network design process. There have also been a number of studies into the implementation of FlexRay. Sections 3.7.1 – 3.7.3 will review the tools and research conducted into the area of FlexRay and the configuration of FlexRay based systems.

3.7.1 Configuration and Monitoring Tools

This section covers a number of tools to configure FlexRay nodes. There are also a number of special tools used to monitor traffic on a FlexRay physical bus. These will be briefly covered in this section.

3.7.1 .1 Vector Informatik Tools

Chapter 13 discusses the Vector CANalyzer software and the VN3600 FlexRay interface hardware module in more detail. CANalyzer uses a hardware interface such as the VN3600 module to monitor traffic on a communication network. The CANalyzer software can display the information in a number of different ways, i.e. graphically or textually. Network data can also be generated for transmission over the network that is under observation. CANalyzer supports a number of different networking protocols

LITERARY REVIEW

such as CAN, MOST and FlexRay. This can help ease the troubleshooting of a system implemented using a number of different networking protocols (Vector Informatik GmbH 2007, p1).

Vector supply a number of different hardware modules to suit different needs and budgets. These include the FlexCard, VN3300 and VN7600 network interfaces as well as the VN3600 hardware interface module. CANoe.FlexRay is a tool for development, simulation and test of ECUs and distributed networks for FlexRay and provides a variant for CAN. As well as these monitor tools Vector supply a number of FlexRay software modules to aid the development of applications (Vector Informatik GmbH 2009a).

Vector offer a FlexRay network development tool called DaVinci Network Designer FlexRay (DaVinci) (Vector Informatik GmbH 2009b). Clusters and controllers can have necessary network constraint parameters assigned and checked against version 2.1 of the FlexRay specification. DaVinci supports the FIBEX file exchange format and provides an interactive method of designing the FlexRay schedule. Figure 3.21 shows the design of a FlexRay schedule using DaVinci (Vector Informatik GmbH 2009b, p1).

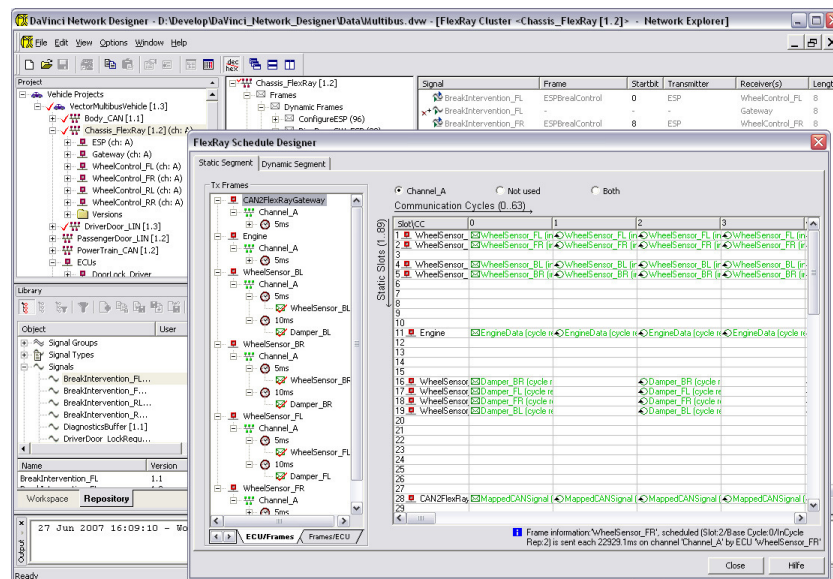


Figure 3.21: DaVinci FlexRay schedule design

3.7.1 .2 Elektrobit Corporation Tools

Elektrobit Corporation (here after referred to as EB) provides a number of tools that support the development of automotive applications. The EB tresos product family is a brand under which all the tools and hardware necessary to develop automotive systems are sold by EB. Figure 3.22 (Elektrobit Corporation 2009a) shows a breakdown of the tresos product family.

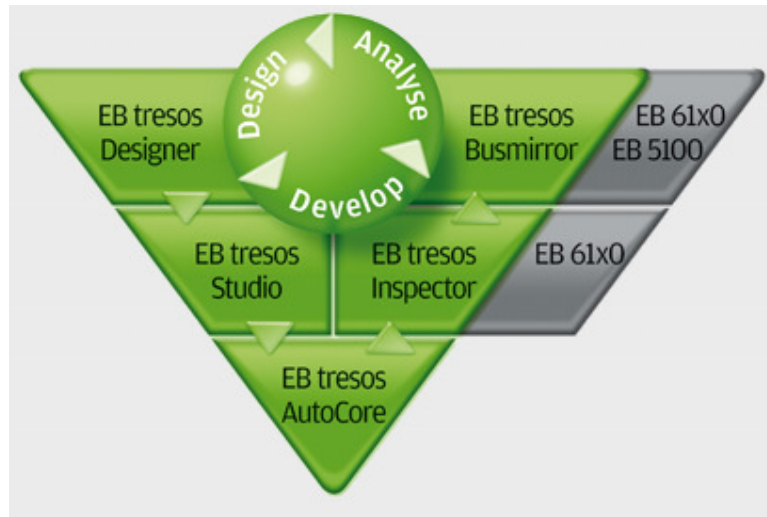


Figure 3.22: EB tresos product family

The EB61x0 and EB5100 are hardware interfaces for both FlexRay and CAN. The EB61x0 is designed to be usable in an automotive application or in an office environment and is designed for a variety of purposes such as the monitoring of a FlexRay bus when used with the tresos Busmirror software. The EB5100 is based on the PMC cards standard with a high performance controller. It is designed to be used with various carrier boards such as PCI, PXI, VME or PHS bus-based systems and to be able to perform calculations to meet real-time system constraints.

EB tresos Designer is a system design tool similar to that of the Vector DaVinci software. Constraints applied to the system in Designer are checked against the FlexRay specification and a FlexRay schedule is designed in an interactive manner. The EB tresos Inspector with the EB61x0 allow for measurement and analysis of FlexRay or CAN networks while EB tresos Busmirror with either the EB61x0 and EB5100 provide FlexRay cluster emulation solutions.

3.7.1.3 dSPACE GmbH Tools

dSpace also provide a number of tools to help support and develop FlexRay systems (dSpace GmbH 2009). The dSPACE FlexRay Configuration Package allows for rapid prototyping and hardware-in-the-loop (HIL) applications. The FlexRay configuration package is used to integrate dSPACE hardware as simulation and/or monitoring nodes in a FlexRay networks. The nodes are configured using the dSPACE FlexRay Configuration Tool using a communication matrix containing schedule information for signals and frames transmitted over a FlexRay bus. This information can also be linked to a MATLAB/Simulink model using the RTI FlexRay Configuration Blockset which results in a FlexRay application that can be executed on a dSPACE based system (dSpace GmbH 2009).

dSpace ControlDesk is a single experiment software package for the development of a controller. The same tools can be used for environment, virtual instrumentation, automation, and parameter set handling. Real-Time data can be recorded and parameters can be tuned using ControlDesk. ControlDesk also provides detailed timing analysis of FlexRay data and can be used with a number of different FlexRay platforms (dSpace GmbH 2009).

3.7.2 FlexRay Design Research

FlexRay is a relatively new communications protocol. There is a lot of interest in using the FlexRay protocol for time critical systems due to the deterministic nature of the TDMA network arbitration scheme. It also provides a dynamic arbitration segment within a communication cycle which provides flexibility and helps reduce redundancy in the networking system. These factors make FlexRay a highly desirable protocol to implement. As was stated at the start of section 3.7, there have been a number of areas where research has been conducted in the area of FlexRay. The research into FlexRay has been done for, amongst other reasons, easing the transition for other protocols such as CAN to this newer communication protocol. This should help reduce redundancy and development time of FlexRay based systems. This section will highlight some of the areas where research has been focused to achieve this.

3.7.2.1 FlexRay PDU Configuration

FlexRay frame length is fixed at the design time of a FlexRay cluster. This will mean that all static frames will transmit a fixed length payload while dynamic frames will adhere to a maximum payload length. It is important then that a frame is configured to transmit the maximum number of data bytes during a communication cycle. To achieve maximum payload usage frames can be split into a number of protocol data units (PDUs). The frame may then contain a number of different messages within the same frame. Any node that is interested in any information contained in one or more of the PDUs must then store the entire frame and extract the desired data.

The paper by Stöger (2008) describes the use of assigning a number of different messages to a single FlexRay frame. Problems associated with this multiplexing of payload data is with the application layer overheads. If a frame is received and there are a number of different PDUs contained within the frame that are desired by the receiving node, then processing time is spent extracting the information and passing the relevant information to the associated tasks. Message PDUs may also be cycle multiplexed within a single frame. For instance a frame may contain 'PDU 1' during every communication cycle but 'PDU 2' is transmitted during every even communication. During all odd cycles 'PDU 3' takes the place of PDU 2 within the transmission frame. This could mean there is the need for this frame to have more than one buffer assigned on a receiving node so that correct PDU extraction can be carried out efficiently. For instance two buffers could be assigned to receive on the frame ID. The first buffer would accept all frames received during even communication cycles and a second buffer would receive during odd communication cycles. This can lead to a more complex view of the communication system but can lead to a more efficient use of buffers and frame transmission.

The paper by Brandstätter and Böke (2008) highlights how PDU-based communication can lead to better FlexRay based applications but also aid in the migration process for older communications schemes such as CAN. They also highlight how development of tools that use the FIBEX file format have aided in networking tools by vendors such as Vector to easily adapt to PDU frame formats. This allows for PDU layer analysis of FlexRay communication and will aid in the transition to this frame assignment procedure.

3.7.2.2 FlexRay Scheduling

Another area that has been researched is the scheduling of frames necessary for time-triggered networks such as FlexRay. The work of Pop et. al. (2003), Pop et. al. (2006), Pop et. al. (2006) and Pop (2007) address this area. In the paper by Pop et. al. (2006) a method to analyse the ‘schedulability’ of the communication protocol is put forward. This is based on the timing properties of the static messages and a worst-case response time for dynamic messages. The timing analysis then determines timing properties for all tasks and messages in a system.

The work of Balogh et. al. (2007) also looks at the scheduling of time-triggered systems such as FlexRay or TTP. This focuses on the allocation of tasks to nodes as well as the scheduling of tasks and the communication parameters. This method should then produce a schedule that incorporates sufficient time to run all tasks and transmit all messages. The configuration of a distributed system can then be optimised to meet a variety of different constraints such as a cost or extensibility constraints.

3.7.2.3 Time Triggered vs. Event Triggered Architectures

A number of studies have been carried out into the suitability of event-triggered and time-triggered protocols. This mainly compares the different protocol types for their suitability to perform different tasks. This can be seen in the paper by Scheler and Schröder-Preikschat (2006) as discussed in section 3.3.

The study and implementation of the different communication protocol variants has also led to the study of the migration process. Event-triggered protocols act in a different manner than time-triggered protocols and this can cause problems when converting from one protocol type to another. Older protocols such as CAN are also in many cases simpler than newer protocols such as FlexRay. The complexity of some of the new protocols, along with the knowledge of the system designers of older protocols, means that the migration from older to newer protocols can be a slow process. The cost to migrate from one protocol to another may also increase when migrating from an older to a newer protocol and this consideration needs to be considered. The increase in cost is associated with training personnel and an increase in costs during development of a new system. Higher cost of new equipment and devices for any new protocol could also be an issue as these tend to be pricier than similar equipment/devices for a more

established protocol. The higher costs can be a contributing factor for a slow or low adoption of a new protocol. This has led to migration frameworks being developed to ease the transition. This can be seen in the work of Cummings (2008).

3.7.3 FlexRay System Development Summary

As can be seen from the tools developed for FlexRay systems and the research that has already been conducted into the area of FlexRay, a lot of effort has been put into the scheduling of the overall system. There has also been a lot of effort put into the analysis of task allocation and ‘schedulability’ across distributed time-triggered systems. This is to optimise a number of constraints such as the number of nodes necessary to fully implement the required system. This analysis can also lead to an optimisation of constraints such as a slot time and number of static slots needed. By using PDUs within a frame maximum utilization of frames can be achieved and a better overall system performance can be configured.

The research outlined in this thesis aims to address a number of questions including what aspect of a node most affects the performance of an automotive distributed system. Proper analysis of the flow of data will allow a designer to identify the bottlenecks with a given system. Configurations of various aspects of the system can then be adjusted to ensure all internal and external node deadlines are met. This will ensure timely and reliable transmission of data. This is an area where little research has previously been conducted. This could be a big advantage where a system designer must improve the performance of a system that incorporates software modules designed by an outside company. Little confidence can be placed in the modification of such software models to streamline the execution as the design may not be known. The system designer must therefore focus efforts, to improve the system, on other aspects of the system.

In the paper by Stöger (2008) as described in section 3.7.2.1, the uses of a number of different message buffers for a single message frame were discussed. This allows PDUs within a FlexRay frame to be multiplexed across a number of different communication cycles. The assignment of more than one message buffers to a single frame will increase have an impact on the resources needed by the node. By analysing the flow of data through a node the message buffer accesses by the host controller may

also be improved. This could lead to an optional assignment of frames to dedicated message buffers and to a FIFO message buffer structure.

3.8 Conclusion

Ageing communication protocols such as CAN with small data package sizes and poor determinism meant that the automotive industry was facing a problem. Master/Slave systems that are available either have small packet sizes or are designed specifically for infotainment systems. The lack of a suitable networking scheme that could handle the requirements for newer, more sophisticated safety systems and x-by-wire technology has led to the development of a new protocol. FlexRay was developed to meet the current and future needs of automotive manufacturers. By using a TDMA scheme a more deterministic system can be achieved. This leads to the possibility of newer and more sophisticated applications being implemented in cars.

With the backing of a number of automotive manufacturers and parts suppliers FlexRay seems set to become a widely used network scheme. It is a new protocol and there is a large amount of research ongoing in various aspects of the protocol. This includes optimisation of the communication cycle. However there are other difficulties that are associated with FlexRay. For Example, the RAM and ROM requirements need to be studied. The implementation costs of communication controllers could be drastically reduced if it was understood what the exact resource requirements are. The overall systems performance of a node could also be improved by fully understanding the flow and timings of data flow through a communication controller.

3.9 References

Ademaj, A., Sivencrona, H., Bauer, G. and Torin, J. (2003) Evaluation of Fault Handling of the Time-Triggered Architecture with Bus and Star Topology, Proceedings of the 2003 International Conference on Dependable Systems and Networks, San Francisco, California, June 22 - 25, 2003, IEEE Computer Society Washington, DC, USA, 123-132.

LITERARY REVIEW

Ahlmark, M. (2000) Local Interconnect Network (LIN) – Packaging and Scheduling, unpublished thesis (M.Eng.), Mälardalen University.

Balogh, A., Pataricza, A. and Rácz, J. (2007) Scheduling of embedded time-triggered systems, Proceedings of the 2007 Workshop on Engineering Fault Tolerant Systems, Dubrovnik, Croatia, ACM, New York, NY, USA.

BMW (2000) byteflight – What is byteflight [online], available: <http://www.byteflight.com/whitepaper/index.html> [accessed 18 November 2008].

BMW Motorrad USA (2008) R1200GS Adventure 2008 Specs [online], available: <http://www.bmwmotorcycles.com/bikes/bike.jsp?b=2008r1200gsa&p=specs&bikeSection=null> [accessed 16 June 2008].

Böhm, P. (2005) Introduction to FlexRay and TTA, Universitas Saraviensis Saarland, Germany.

Brandstätter, W. and Böke, C. (2008) AUTOSAR PDUs Conquer FlexRay, Automotive Special Edition FlexRay, Hanser, 24-26.

CiA (2007) Controller Area Network (CAN) [online], available: <http://www.can-cia.org/> [accessed 30 Oct 2007].

Cummings, R. (2008) Easing the Transition of System Designs from CAN to FlexRay, April 14-17 2008, Detroit Michigan, USA, SAE International, Warrendale, Pennsylvania, USA.

Denner, V., Maier, J. Kraft, D. and Spreitz, G. (2004) Data Processing and Communication Networks in Motor Vehicles, Bosch Automotive Handbook, Plochingen, Robert Bosch GmbH, 1064-1076.

LITERARY REVIEW

Dependable Computer Systems (2007) DESIGNER PRO 4.3.0 - DESIGNER PRO, DESIGNER PRO <LIGHT> and DESIGNER PRO <SYSTEM> Document Version 2.2, Vienna, Austria.

dSpace GmbH (2009) dSPACE Products for FlexRay Applications [online], available at:http://www.dspace.com/ww/en/pub/home/products/our_solutions_for/flexray_development_with_ds/dspace_products_flexray_applic.cfm?nv=n2 [accessed 15 May 2009].

Ecartec Ltd. (2008) CAN Bus Explained [online], available: http://ecartec.com/can_bus_explained.html [accessed 27 June 2008].

Elektrobit Corporation (2009a) EB tresos[®]: The Product Family for the Development of ECU Software [online], available: <http://www.elektrobit.com/index.php?888> [accessed 13 May 2009].

Elektrobit Corporation (2009b) EB 61x0: Powerful FlexRay and CAN Bus Interface Hardware [online], available: http://www.elektrobit.com/what_we_deliver/automotive_software/products/eb_tresos_-_ecu_software_development/eb_61x0 [accessed 13 May 2009].

Elektrobit Corporation (2009c) EB 5100: Active FlexRay and CAN Interface Hardware for PCI or PXI [online], available: http://www.elektrobit.com/what_we_deliver/automotive_software/products/eb_tresos_-_ecu_software_development/eb_5100 [accessed 13 May 2009].

Elmenreich, W. and Ipp, R. (2003) Introduction to TTP/C and TTP/A, Workshop on Time-Triggered and Real-Time Communication Systems, 12 February, Manno, Switzerland.

Elmenreich, W. and Krywult, S. (2005) A Comparison of Fieldbus Protocols: Lin 1.3, Lin 2.0 and TTP/A, Proceedings of the 10th IEEE International Conference on Emerging Technologies and Factory Automation, Catania, Italy, 19 -22 September, IEEE Computer Society Washington, DC, USA, 747 – 753.

LITERARY REVIEW

FlexRay Consortium (2005) FlexRay Communication System Protocol Specification, Version 2.1 Revision A, Stuttgart: FlexRay Consortium GbR.

FlexRay Consortium (2007) about FlexRay [online], available:

<http://www.flexray.com/index.php?sid=254188fe2bd59eb7108227f0adea90f5&pid=80&lang=de> [accessed 19 Oct 2007].

Fujitsu (2008) Applications – Body and Comfort Electronics [online], available:

<http://www.fujitsu.com/emea/services/industries/automotive/microelectronics/applications.html> [accessed 12 June 2008].

Heller, C., Schalk, J., Schneele, S. and Reichel, R. (2008) Approaching the Limits of FlexRay, The 7th IEEE International Symposium on Network Computing and Applications, Cambridge, Massachusetts, USA, July 10 - 12, IEEE Computer Society Washington, DC, USA, 205 -210.

Jurgen, R. K. (1999) Automotive Electronics Handbook, 2nd edition, USA, McGraw-Hill.

Kopetz, H. (1991) Event-Triggered Versus Time-Triggered Real-Time Systems, Lecture Notes In Computer Science; Vol. 563, Proceedings of the International Workshop on Operating Systems of the 90s and Beyond, Dagstuhl Castle, Germany, July 8-12, Springer-Verlag London, UK, 87 – 101.

Kopetz, H. (2000) Time-Triggered Architecture, International Federation for Information Processing WG 10.4 on Dependable Computing And Fault Tolerance, January 22-23.

Kopetz, H. and Grunsteidl, G. (1993) TTP - A Time-Triggered Protocol for Fault-Tolerant Real-Time Systems, Digest of Papers of The Twenty-Third International Symposium on Fault-Tolerant Computing, Toulouse, France, June 22-24, IEEE Computer Society Washington, DC, USA, 524 – 533.

LITERARY REVIEW

Leen, G. and Heffernan, D. (2002) Expanding Automotive Electronic Systems, Computer, 35(1), 88-93

LIN Consortium (2006) LIN Specification Package Revision 2.1, Stuttgart: LIN Consortium.

MOST Cooperation (2006) MOST Specification, Revision 2.5, Karlsruhe: MOST Cooperation.

Navet, N., Song, Y., Simonot-Lion, F. and Wilwert, C. (2005) Trends in Automotive Communication Systems, Proceedings of the IEEE, vol. 93, No. 6, IEEE Computer Society Washington, DC, USA, 1204 – 1223.

Pop T. (2007) Analysis and Optimisation of Distributed Embedded Systems with Heterogeneous Scheduling Policies, unpublished thesis (PhD.), Linköpings Universitet.

Pop, P., Eles, P. and Peng, Z. (2003) Schedulability Analysis and Optimisation for the Synthesis of Multi-Cluster Distributed Embedded Systems, IEE Proceedings - Computers & Digital Techniques, Vol. 150, Issue 5, Sept. 2003, pp. 303-312.

Pop, T., Pop, P., Eles, P. and Peng, Z. (2007) Bus Access Optimisation for FlexRay-Based Distributed Embedded Systems, Proceedings of the Conference on Design, Automation and Test in Europe, Nice, France, April 16-20 2007, IEEE Computer Society Washington, DC, 51 – 56.

Pop, T., Pop, P., Eles, P., Peng, Z. and Andrei, A. (2006) Timing Analysis of the FlexRay Communication Protocol, in Proceedings of the 18th Euromicro Conference on Real-Time Systems, Dresden, Germany, July 5-7, 2006, IEEE Computer Society Washington, DC, USA, 203-216.

Robert Bosch GmbH (1991) CAN Specification, Version 2.0. Reutlingen: Robert Bosch GmbH.

LITERARY REVIEW

Scheler, F., and Schröder-Preikschat, W. (2006) Time-Triggered vs. Event-Triggered: A matter of configuration?, Proceedings of GI/ITG Workshop on Non-Functional Properties of Embedded Systems, Nuremberg 27-29 March, VDE Verlag, Berlin, 107 - 112.

Schofield, M. (2006) Controller Area Network [online], available: <http://www.mjschofield.com/index.htm> [accessed 30 Oct 2007].

Softing (2008) CAN Bus Arbitration Method [online], available: <http://www.softing.com/home/en/industrial-automation/products/can-bus/more-can-bus/communication/bus-arbitration-method.php> [accessed 17 June 2008].

STMicroelectronics (2001) AN1278 Application Note - Lin (Local Interconnect Network) Solutions, Geneva: STMicroelectronics.

Stöger G. (2008) Creating FlexRay COM Stack Configurations for ECUs in Complex Networks, Automotive Special Edition FlexRay, Hanser, 20-23.

TechInsights (2008) Consumer and automotive electronics converge: Part 2 - A MOST implementation [online], available: <http://www.automotivedesignline.com/howto/198001031> [accessed 16 June 2008].

TTA-Group (2004) TTP – Easy to Read, Vienna, Austria: TTA-Group.

TTA-Group (2008) TTA-Group – News [online], available: <http://www.ttagroup.org/news/pressreleases.htm> [accessed 17 November 2008].

Vector Informatik GmbH (2006) FlexRay Protocol Reference Chart, Stuttgart, Vector Informatik GmbH.

Vector Informatik GmbH (2007) CANalyzer 7.0 Datasheet, Stuttgart, Germany.

LITERARY REVIEW

Vector Informatik GmbH (2008) Solutions for LIN, version 3.4, Stuttgart, Vector Informatik GmbH.

Vector Informatik GmbH (2009a) Solutions for FlexRay Networking [online], available at: http://www.vector.com/vi_flexray_solutions_en.html [accessed 15 May 2009].

Vector Informatik GmbH (2009b) DaVinci Network Designer FlexRay v2.2, Stuttgart, Vector Informatik GmbH.

Vienna University of Technology – Real-Time Systems Group (1997) The TTP Protocols [online], available: <http://www.vmars.tuwien.ac.at/projects/ttp/ttpmain.html> [accessed 17 June 2008].

Chapter 4 . FlexRay

4.1 Introduction

The FlexRay protocol was developed when BMW and DaimlerChrysler decided to cooperate. Together they had realised that automotive network solutions at the time were inadequate for future developments such as drive-by wire. They were soon joined by Motorola and Philips to form the FlexRay consortium. Other leading automotive and electronic companies such as Bosch and VW soon joined (FlexRay Consortium 2007). The partnership soon led to a protocol specification which is the basis for FlexRay systems. This protocol has also been the basis of FlexRay IP-modules such as the Bosch E-Ray communications controller (Robert Bosch GmbH 2007).

The core of the FlexRay protocol is a time-triggered communication system. This is in contrast to some earlier event-triggered automotive applications such as CAN. The use of a time-triggered protocol ensures a fixed delay in the transmission of data. This is in contrast to an undeterminable time that data must wait before it is transmitted using an event-triggered protocol, due to there being potentially higher priority messages blocking access to the bus. The FlexRay approach is more suitable for safety applications such as brake-by-wire when it is important that a message is not blocked from accessing the network.

The protocol also provides flexibility and determinism by providing a dynamic segment in a communication cycle. In this way it provides both synchronous and asynchronous communication modes as standard. The physical layer also includes a bus guardian to support error containment and provides for a data rate of up to 10Mbit/sec on each of two channels giving an equivalent overall data rate of up to 20Mbit/sec. FlexRay can therefore be seen as being designed for present and future needs of automotive applications (FlexRay Consortium 2007).

If not otherwise stated, the information in this chapter was referenced from the FlexRay Consortium (2005).

4.2 Network Topology

The FlexRay protocol allows for various bus topologies. These can be a point to point connection, a passive star, linear passive bus, active star network, cascaded active stars, hybrid topologies and dual channel topologies (FlexRay Consortium 2005, p.21; FlexRay Consortium 2006, pp26-31).

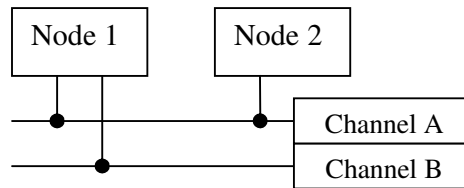


Figure 4.1: A passive bus topology

Figure 4.1 and Figure 4.2 show the basic layout of a passive bus topology and a single channel hybrid network respectively. Note that in a network a node need not be attached to both channels of the network, also a node attached to a single channel need not be attached to channel A but to either channel A or channel B. The FlexRay protocol will support hybrid topologies as long as the limits of each topology which makes up the hybrid topology (i.e. the star and bus topologies) are not exceeded.

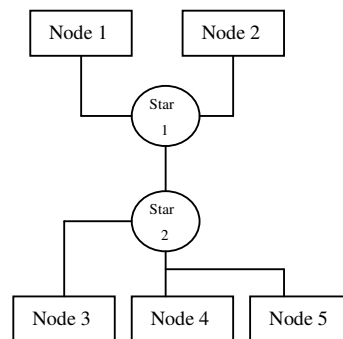


Figure 4.2: Single channel hybrid network

It should also be noted that each channel can be implemented as a different bus topology. For instance channel A can be a bus topology while channel B is implemented as a star topology. This makes FlexRay a very flexible and adaptive communications system for a wide range of applications.

4.3 FlexRay Hardware

Each FlexRay node has a communication controller, a host, a power supply unit and two bus drivers, one for each channel. Figure 4.3 (FlexRay Consortium 2005, p26) shows the logical connections of each element.

The host handles the applications of the system while the FlexRay protocol is handled by the communications controller. The bus driver is used to read and write data to the physical medium over which the data is transmitted. In sleep mode it also has the ability to start a wakeup procedure if it detects a wakeup signal. The communications controller will mainly handle the framing of data and the checking of received data. This is to ensure no data was corrupted before passing it to the host.

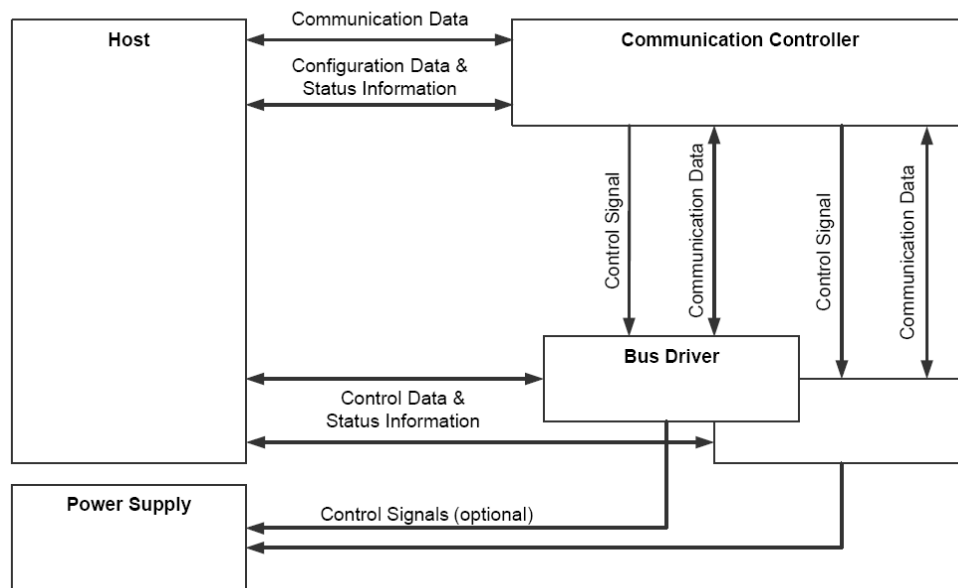


Figure 4.3: Logical interface

The host passes information such as control information and payload data to the communications controller. The communication controller relays status information and data received. The host interface to the bus driver allows it to change the operation of the bus driver as well as read status and error flags.

The connections between the communications controller and the bus driver allow data to be transferred from the communications controller to the bus driver and vice versa. There is also a 'transmit enable not' line which indicates that the bus driver can transmit data on its corresponding channel.

4.4 Global Time and Timing

Figure 4.4 (FlexRay Consortium 2005, p170) shows the timing hierarchy used in FlexRay. It consists of a communication cycle, macrotick and microtick levels. These will be discussed from the bottom up.

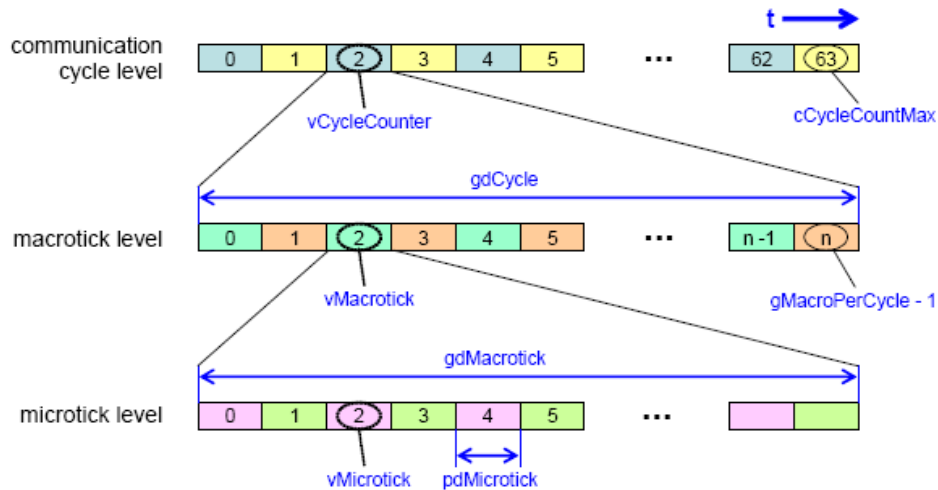


Figure 4.4: Timing hierarchy

4.4.1 Microtick

In a FlexRay system the most basic unit of time is a microtick. This is derived from a node's local oscillator. In this way the length of a microtick will vary from controller to controller. This leads to controllers drifting away from each other with respect to the beginning of segments of the communication cycle. This can lead to errors and as such there needs to be a way to synchronise time.

4.4.2 Macrotick

The macrotick in a FlexRay system is made up of fixed number of microticks. The number of microticks which make up a macrotick may vary between nodes in a system due to different operating frequencies. This will ensure that all macroticks will have the same duration across the network within a given tolerance. This means that all

nodes on a network will have the same number of microticks per communication cycle. The communication cycle will be covered in section 4.5.

4.4.3 Global Time

In a FlexRay network it has been established that there are different levels of time representation to help the network stay synchronised. However this will still lead to a drift of the local time of the nodes if there is no correction applied to the nodes. This is because there is no global reference point for the time. Instead there is a local time for each node. This is the controller's idea of the global time based on aspects such as its idea of what microtick was last transitioned and when it should transition for the next microtick. Every node uses a synchronisation algorithm to keep its view of the global time as accurate as possible.

4.4.4 Synchronisation Algorithm

This is a distributed clock synchronisation algorithm, where all the nodes in a cluster synchronise themselves to the other nodes in the cluster by monitoring the transmissions of sync frames sent from other nodes. A node will then try to adjust its view of the global time to that of the other nodes. After this process has been carried out all nodes should share the same view of the global time to within a given tolerance. This tolerance is known as the precision of the network.

The clock synchronisation is performed using two processes. These are the microtick generation process and the clock synchronisation process.

4.4.4.1 Clock Synchronisation Process

This process measures both rate and offset differences of the expected times of arriving messages and the actual arrival times. Rate correction is done during the whole communications cycle while offset correction is done during the network idle time. The correction values are in terms of microticks which are needed to be added to the communications cycle and this value may be negative as well as positive.

The calculation of the offset correction is done every cycle but the corrections are only applied during the idle time of odd communication cycles. The calculation

must be completed before the offset correction phase begins but any calculated values must not be applied until the idle time.

The rate correction values are calculated once every two cycles after the static section of the odd cycle. The values are based on the values observed during the two cycles before the calculation. Again the calculation must be complete before the offset correction phase begins but any calculated values must not be applied until the idle time. Figure 4.5 (FlexRay Consortium 2005, p172) shows the relationship between clock synchronisation and the media access timeframe.

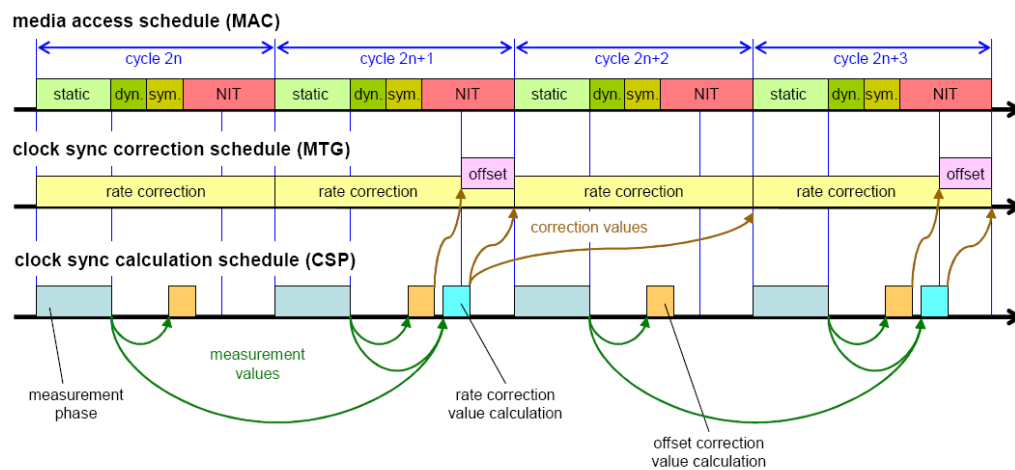


Figure 4.5: The relationship between clock synchronisation and the media access time frame

4.4.4.2 Macrotick Generation Process

This process produces macroticks which are ‘corrected’ based on the rate and offset correction values.

4.4.5 Correction Term Calculation

In order to calculate the correction value a fault-tolerant midpoint algorithm is used. This defines a parameter ‘k’ based on the number of terms in a sorted list of time deviations. These deviations are between the reference points for expected timestamps and actual timestamps. Table 4.1 (FlexRay Consortium 2005, p184) displays the relationship between the number of values in the sorted list and ‘k’.

LITERARY REVIEW

Number of values	k
1-2	0
3-7	1
>7	2

Table 4.1: k as a function of a list of values

The value of k is obtained from this table and it is then used to figure out how many of the largest and smallest values should be removed, i.e. if k is calculated to be 2 then the two largest and smallest values are taken out of the list. The next largest and smallest are then averaged and the result is the node's deviation from the global time for the purposes of the correction.

The calculated values will then be checked against predefined limits. If each of the values lies within its limits then the node is said to be synchronised. Otherwise an error condition is detected and appropriate flags are set or a procedure can be put in place to change the correction term to its limit or another predefined value.

4.5 Media Access Control

Figure 4.6 (FlexRay Consortium 2005, p100) shows the breakdown of the communication cycle into the various segments. The segments as shown in Figure 4.6 will be discussed from left to right.

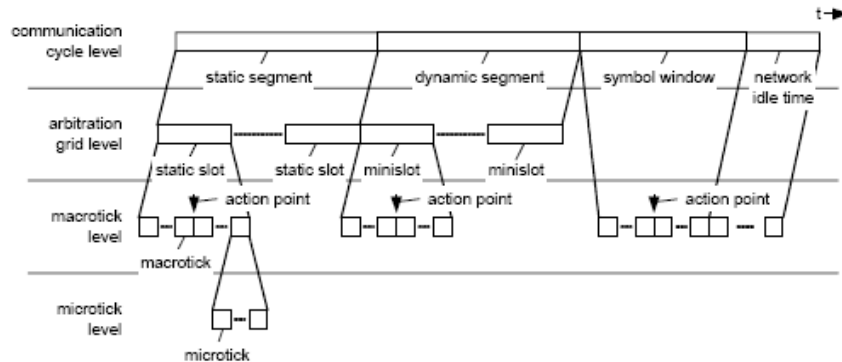


Figure 4.6: Communication cycle

4.5.1 Static Segment

The static segment transmits data using a Time-Division Multiple Access (TDMA) technique to allow different nodes to transmit and receive data over the network at predefined times.

The static segment is broken down into smaller time slots known as static slots. These slots are assigned to a message ID so that only that message may be sent during that slot time every communication cycle. There is a possibility to use a cycle multiplexing system however.

During the transmissions of frames in the static segment frames may be sent over one or both of the channels at a time. Only one node however can transmit on a given channel with a given frame ID during a given slot.

As has been stated the entire network shares a 'global view' of the time on a given network. This ensures that nodes on the network agree on when a slot starts and ends. This helps to avoid different messages being sent out at the same time.

4.5.2 Dynamic Segment

To make FlexRay more usable there can be a dynamic segment included if desired by the network designer. This is where a node can transmit data at arbitrary times. If two nodes want to transmit data at the same time then the message with the lower message ID is transmitted first and the other messages have to wait until that message is transmitted before commencing transmission. This is similar to CAN, but transmission can only begin if there is time to transmit the entire message before the end of the dynamic segment. If there is insufficient time left to transmit the message then the message will be kept for the dynamic segment of the following communication cycle.

The dynamic segment is broken up into smaller sections known as minislots. These are defined in terms of macroticks where the start of a minislot defines an action point where transmission may begin.

During the Dynamic segment the slot counters may be incremented at different action points and thus two different message IDs may be transmitted on the bus at the same time over the two channels. The dynamic messages' slot IDs are numbered sequentially from the last static message ID.

4.5.3 Symbol Window

A symbol is used to signal a need to wakeup a cluster amongst other things. The meaning of a symbol depends on the symbol sent and the status of the controller at the time. Within the symbol window a single symbol may be sent. If there is more than one symbol to be sent then a higher level protocol must determine which symbol gets priority as the FlexRay protocol provides no arbitration for the symbol window.

4.5.4 Network Idle Time

The network idle time is used to calculate clock adjustments and correct the nodes' view of the global time. It also performs communication specific tasks and uses the remaining time of the communication cycle.

4.6 Frame Format

Figure 4.7 (FlexRay Consortium 2005, p90) shows the frame format of a FlexRay message. It is broken down into three sections: the header, payload and trailer sections.

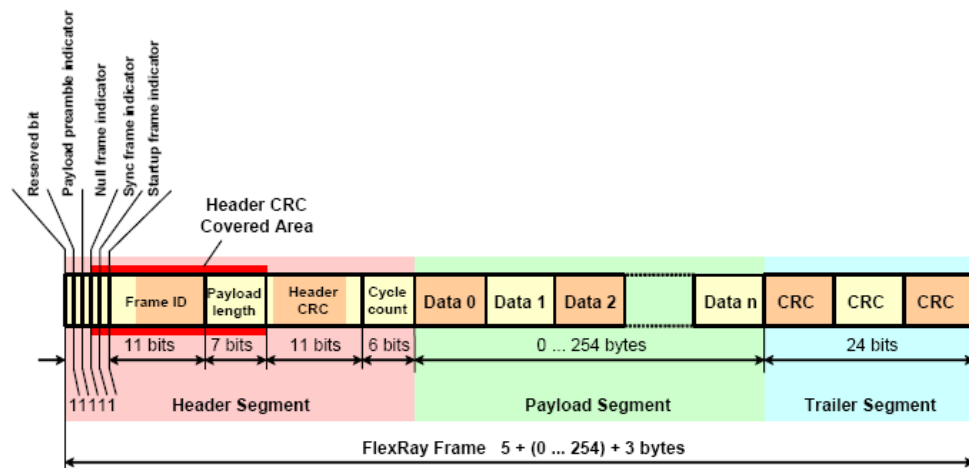


Figure 4.7: Frame format

The frame bits are transmitted from left to right as you look at Figure 4.7, i.e. the reserved bit is sent first followed by the payload preamble indicator bit etc.

4.6.1 Header Section

The header section is broken down into smaller sections. It is five bytes in length which is broken down into a reserved bit, payload preamble indicator bit, null frame indicator bit, sync frame indicator bit, startup frame indicator bit, a frame ID (11 bits), the payload length (7 bits), a header CRC (11 bits) and a cycle count (6 bits). The CRC is not computed by the communications controller which is transmitting the frame. Instead the CRC is passed to the communications controller by the host as it does not generally change during the static segment. The CRC is recalculated by a receiving communications controller. This is to ensure that a received frame was received with no errors. This CRC code is calculated for all channels and uses the following polynomial: $x^{11} + x^9 + x^8 + x^7 + x^2 + 1$. The initialised value for the register that is used to calculate the CRC is the same for both channels and is 0x1A.

For further information on how the CRC is generated and the other sections of the header see the FlexRay Consortium protocol specification (2005, pp97-99).

4.6.2 Payload Section

The payload section is used to send data and contains 0 to 254 bytes of data. Each byte of data is generally referred to by its position in the payload, i.e. the first byte is called “data 0”, the second “data 1” and so on.

In some cases the payload may also be used to transmit more frame information as an option. This data would be a message ID field in the dynamic segment and a network management vector in the static segment.

4.6.2.1 Network Management Vector

This can take up 0 to 12 bytes of the payload section and would be placed at the start of this section i.e. “nm0” would be used instead of “data 0” and “nm1” instead of “data 1” etc.

In order to allow a node to determine if a message contains a network management vector the network preamble indicator bit is set in the header section and it must only be transmitted during the static segment of the communications cycle. All nodes in a cluster must be configured with the same network management vector length.

LITERARY REVIEW

The network management vector is used to coordinate startup and shutdown decisions based on factors such as the application state. It is part of a network management service.

4.6.2.2 Message ID Field

During the dynamic segment of the communications cycle a message ID field may be placed as the first two bytes in the payload section. This allows the receiving frame to determine how the data should be used or filtered. The message ID is 16 bits long and can only be transmitted during the dynamic segment of the communication cycle. To determine whether a message contains a message ID a receiver checks the payload preamble indicator bit in the header. If this is set the payload contains a message ID field.

4.6.3 Trailer Section

The trailer section is made up of a 24 bit CRC code for the frame (FlexRay Consortium 2005, p96). It is calculated over the header and payload sections of the frame and the polynomial used for all channels is:

$x^{24} + x^{22} + x^{20} + x^{19} + x^{16} + x^{14} + x^{13} + x^{11} + x^{10} + x^8 + x^7 + x^6 + x^3 + x + 1$
This will give a Hamming distance of six for a payload of up to 248 bytes, otherwise for payloads of 248 bytes and over the Hamming distance is four.

The initial value of the register used to calculate the CRC is different depending on which channel is being used. For channel A the value is 0xFEDCBA and for channel B the value is 0xABCDEF. The CRC for the frame is calculated, unlike the header CRC, by the communications controller. This means that the frame CRC is calculated by the communications controller during transmission and reception of a frame.

On reception of a frame, the transmitted CRC is checked against a CRC which is calculated based on the received header and payload sections. If these two values differ then an error has been detected, otherwise the frame was received error free. The result of this should be signalled to the host by using an indicator such as a flag. The host can then follow an error procedure. This could involve signalling to the network that a frame was received with an error or the host could attempt to recover the data depending on the configuration of the system.

4.7 Coding & Decoding

As there are two channels there is a need to perform coding and decoding independently, however it is carried out in the same manner. In order to implement the coding and decoding FlexRay implements three processes, the coding/decoding process (CODEC), the bit strobing process and the wakeup pattern process.

4.7.1 Bit Stream Assembly

To transmit a frame the following steps need to be taken:

1. The frame data is broken up into individual bytes.
2. A transmit start sequence followed by the frame start sequence is transmitted.
3. An expanded byte sequence for each data byte is created by prefixing the byte start sequence before the bits of the bytes.
4. This is then assembled, in order, into a single bit stream for transmission.
5. The CRC is then calculated for the frame, and expanded byte sequences are created for this data before being appended to the bit stream.
6. The frame end sequence at the end of the bit stream is added.
7. If the frame is in the dynamic segment the dynamic trailing sequence is appended.

Figure 4.8 (FlexRay Consortium 2005, p57) shows a bit stream with all encoding having been done in the static segment. For a dynamic segment diagram see the FlexRay Consortium (2005, p58).

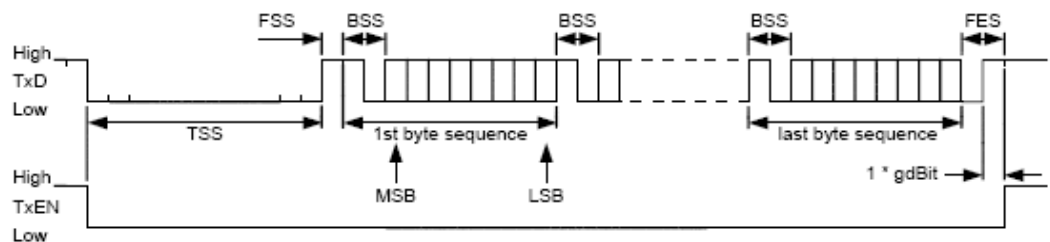


Figure 4.8: Encoded bit stream

4.7.2 Frame Encoding

In order to transmit data a node must represent the communication elements as a bit stream before it can be transmitted over the physical medium. This section deals with how a frame is encoded for transmission.

4.7.2.1 Transmission Start Sequence (TSS)

This is used to ensure proper setup of the network. An active star will use this to properly configure input and output connections. This type of setup will cause an active star to truncate a number of bits at the start of a frame or symbol. This will therefore ensure that the frame or symbol contents are not corrupted or truncated.

4.7.2.2 Frame Start Sequence (FSS)

The FSS is used to compensate for possible quantisation errors after the TSS. It is defined as a high bit.

4.7.2.3 Byte Start Sequence (BSS)

This sequence is used for timing information of the streaming bits. It consists of a high bit followed by a low bit. Each frame data byte will be sent onto the channel as an expanded byte sequence, where eight data bits are prefixed with a single byte start sequence.

4.7.2.4 Frame End Sequence (FES)

This end sequence is used to mark the last byte of a frame. It is a low bit followed by a high bit. It is appended to the last expanded byte sequence of the frame. These are the last two bits sent if the frame is transmitted in the static segment. If this is the case the transmit enable line will be set to high to prevent further transmission. For a frame sent in the dynamic segment there is an additional sequence added.

4.7.2.5 Dynamic Trailing Sequence (DTS)

The DTS is used for frames sent in the dynamic segment only. It is so that the exact minislot action point can be determined and to prevent false detection of a channel idle state by receiving nodes. This is transmitted directly after the frame end sequence.

It consists of a low level transmission of at least one bit length, but the length is not fixed for longer periods. This is followed by a high output for one bit length. Once the output has been high for one bit time the transmit enable line is set high. This will mean that the duration of the dynamic trailing sequence is variable and can range in length between two bits and the length of a minislot plus two bit times.

4.7.3 Frame Decoding

This section deals with the decoding process of received frames on a channel. This is again performed on each channel in the same manner but separately. The decoding of a frame or a symbol is carried out one at a time i.e. if a frame is being decoded another frame or symbol can not be decoded at the same time on the channel. The successful decoding of a frame/symbol will happen as long as at least the channel idle delimiter time is observed between the last bit of the previous frame/symbol and the current frame/symbol. A successfully decoded frame or symbol will not guarantee that the received data is correct or valid. Figure 4.9 (FlexRay Consortium 2005, p66) shows the frame decoding process.

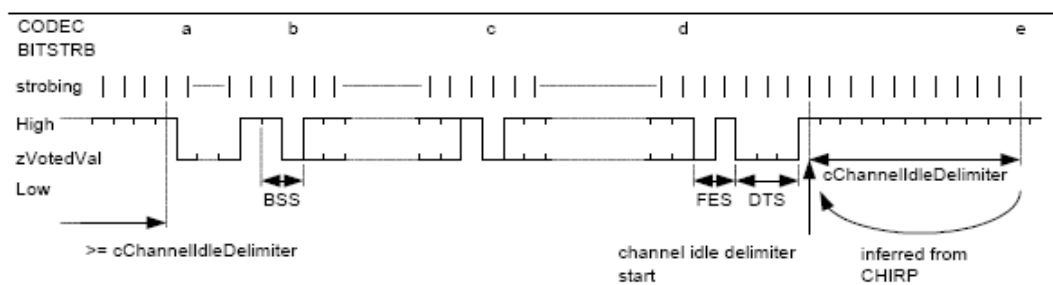


Figure 4.9: Received bit stream

At each of the points in the diagram (a-e) the following is happening:

- The end of the channel idle point is detected.
- A potential frame start sequence is detected.
- The header is received.

LITERARY REVIEW

- d. All frame data is received at this stage and the frame ending sequence is expected followed by a dynamic trailing sequence if the frame was sent during the dynamic segment.
- e. The channel idle delimiter time is reached and another frame or symbol can be received.

4.7.4 Symbol Encoding

There are three defined symbols used in the FlexRay protocol. These are:

1. The collision avoidance symbol (CAS).
2. The media access test symbol (MTS).
3. The wakeup symbol (WUS).

The symbols for the CAS and MTS use the same bit pattern and are distinguished by the receiving node based on the status of the node. The encoding process does not distinguish between them.

4.7.4.1 CAS and MTS

The CAS symbol is used by coldstart nodes to begin startup of a cluster while the MTS is used for testing the media access control operation. The bit pattern for the CAS and MTS is as follows:

1. A transmission start sequence is first transmitted.
2. A low level is transmitted for a defined symbol period.

The symbols are transmitted with the transmit enable being synchronous with the transmit data signal. This is shown in figure 4.10 (FlexRay Consortium 2005, p59).

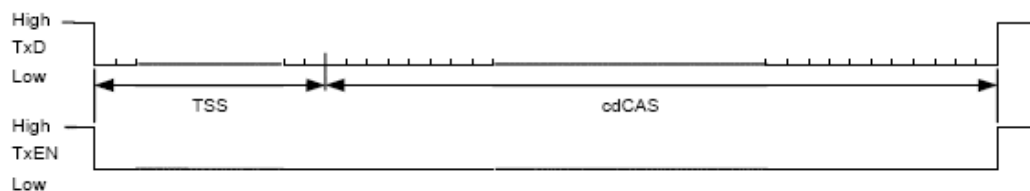


Figure 4.10: CAS and MTS encoding

4.7.4.2 WUS

The WUS symbol is used to signal to other nodes on the cluster a desire to wakeup the network and to begin transmission of frames. The node shall transmit a low logic level for a given ‘wakeup low’ period. This is followed by an idle state which has a defined time. This will then be repeated for a globally defined number of times. Figure 4.11 (FlexRay Consortium 2005, p59), shows a wakeup pattern made up of two wakeup symbols.

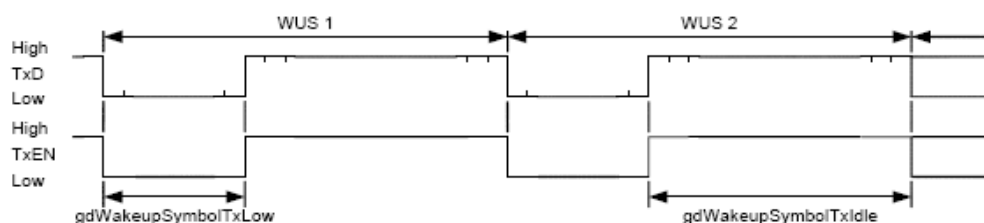


Figure 4.11: Wakeup pattern using two wakeup symbols

A node should be able to detect any transmissions on a channel during transmission of a wakeup pattern in case there is another wakeup pattern from another node or activity already on the bus. This sort of collision can then be handled to ensure that there is no error or protocol violation performed on the channel.

4.7.5 Symbol Decoding

4.7.5.1 CAS and MTS

The received symbol will be decoded by the node in the same way for both symbols.

As the transmission start sequence is a low level for a given time and this is immediately followed by the CAS or MTS symbol, which is also represented by a low level of a given time, there is no way for the receiver to distinguish between the symbol and the start sequence. Therefore a successful detection of a symbol is determined if a low level is detected for a given time within the CAS/MTS min and max limits defined in the protocol of the nodes of the network.

4.7.5.2 WUS

The detection of a wakeup pattern is to be considered as being successful under the following conditions:

1. A low level that is as long as the WUS low period is detected.
2. This is followed by a high level that has duration of the WUS idle time.
3. Steps 1 and 2 are repeated until the number of wakeup symbols which make up a wakeup pattern are received
4. The duration of the wakeup pattern does not exceed its constraint limit.

4.7.6 Sampling and Voting

When data is sent on a channel, nodes which receive the data must determine what was sent on the channel. In order to do this a sampling and majority voting scheme is used. This is done independently on each channel.

The sampling is done at the received input and each sample is stored. The sampling period and number of stored samples depends on the application and hardware used. The node shall then perform a majority voting operation on the stored data.

This majority voting operation is used to filter any glitches detected on the channel. In this case a glitch is an event which temporarily changes the logic value of the received data to that of a value other than that which was transmitted. The receiving node shall continually check the stored samples and if the majority of the samples are a logic one then the output from this process shall be a logic one. Otherwise a logic zero is detected. This voted value is the value which is then used by further decoding processes or stored as the received message.

It should be noted that this process will cause a delay to appear in the received bit pattern or the voted value which is relative to the clock period of the sample clock. Figure 4.12 (FlexRay Consortium 2005, p61) shows a received bit pattern along with a glitch and the delay caused by this process.

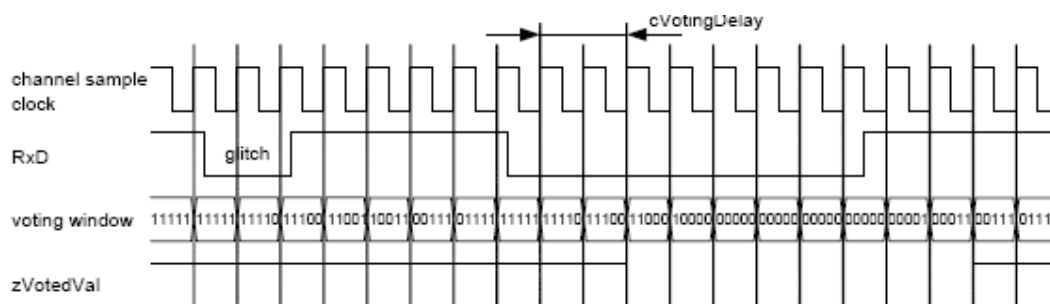


Figure 4.12: Sampling and majority voting of a received bit pattern at the input

The example shown in Figure 4.12 shows a sample length of 5 and sampling is done on the rising edge of the clock.

4.8 Wakeup

This section covers the basics of getting a FlexRay cluster to full operation from the sleep mode.

4.8.1 Cluster Wakeup

The cluster wakeup is performed by a macro and follows the procedure outlined below. It is necessary that the bus drivers are supplied with power. If the bus driver is supplied with power it has the ability to wake up the other nodes' systems. There must also be a wakeup source supplied to at least one node.

The host can transmit the wakeup pattern on each of its channels individually but it should not be transmitted on both channels at the same time to avoid faulty nodes interrupting communication on both channels. The host will configure which channel is to be woken up and ensure communication on the channel is not disturbed. The protocol also allows for nodes connected to a single channel to wakeup the network on both channels. This is done through a node connected to both channels being used to wakeup the other channel. To avoid certain failures it is recommended that both channels should be woken by different controllers.

If the wakeup pattern is successfully received by a node which is asleep, this node shall wakeup. The bus driver will handle the detection of the wakeup pattern, with the communications controller only needing to recognise the wakeup pattern

during the wakeup and startup phases to avoid collisions. It is also impossible for the communications controller to determine if all nodes connected to a network received the wakeup pattern and are awake at the startup phase.

It should be noted that any number of nodes trying to wakeup the network will be resolved by the wakeup procedure so that only one node will wakeup the network. However if there is a fault which causes two nodes to transmit the pattern at the same time, then the resulting signal can still wakeup the network.

The Bosch E-Ray chip fully supports the FlexRay protocol and an application note has been produced on the wake up procedure (Robert Bosch GmbH 2006) that is a good reference on the requirements to wakeup a node.

4.8.2 Startup and Reintegration

To communicate across a TDMA system there has to be synchronisation of all of the nodes. A startup procedure is therefore put in place to initially synchronise all the nodes.

To start up a network all the nodes must first be awake. When all the nodes are ready then a startup process or 'coldstart' can begin. This is done by a few coldstart nodes. There is a limited amount of coldstart nodes in a network. In a network of less than three nodes, all nodes are configured to be coldstart nodes. For networks with three or more nodes, there must be at least three nodes configured as coldstart nodes.

To begin the startup procedure, a coldstart node transmits a CAS. It can then transmit frames. After the first four cycles following the CAS it is joined by the other nodes, starting with the coldstart nodes then the remaining nodes in the network. All frames sent during startup are sync frames and so all coldstart nodes should be configured as sync nodes.

After collecting startup frames, if there are no clock correction errors detected then a node will enter normal operation. This process varies depending on the configuration of the node. For further detail see the FlexRay Consortium protocol (2005, p157).

4.9 Conclusion

The protocol outlined by the FlexRay consortium has been discussed in this document. It has briefly covered basics of why the protocol is needed and how it is implemented.

As can be seen it was developed with current and future needs in mind. However FlexRay is still a new technology. As such there are areas where improvement may be gained or needed. This will become clearer as more and more vehicles have FlexRay systems implemented on them. The first car to do so was the 2006 X5 (Berwanger et al. 2004; BMW Manufacturing Co. 2006) with more vehicles expected in 2009. As the technology matures the use of FlexRay is set to increase in areas such as drive-by-wire and safety systems. This makes FlexRay a very good research and development area.

4.10 References

Berwanger, J., Schedl, A. and Peller, M (2004) BMW- First Series Cars with FlexRay in 2006, Automotive electronics + systems, Development Solutions 19 for FlexRay ECUs, 6-8.

BMW Manufacturing Co. (2006) THE NEW BMW X5

Perfect Blend of Driving Dynamics, Functionality and Exclusivity [press release], 8 August, available:

http://www.bmwusfactory.com/media_center/releases/release.asp?intReleaseNum=209&strYear=2006 [accessed 2 October 2007].

FlexRay Consortium (2005) FlexRay Communication System Protocol Specification, Version 2.1 Revision A, Stuttgart: FlexRay Consortium GbR.

FlexRay Consortium (2006) FlexRay Communications System Electrical Physical Layer Specification, Version 2.1 Revision B, Stuttgart: FlexRay Consortium GbR.

LITERARY REVIEW

FlexRay Consortium (2007) about FlexRay [online], available:

<http://www.flexray.com/index.php?sid=254188fe2bd59eb7108227f0adea90f5&pid=80&lang=de> [accessed 19 Oct 2007].

Robert Bosch GmbH (2006) E-Ray Application Note AN001 Wakeup, Revision 1.0, Reutlingen: Robert Bosch GmbH.

Robert Bosch GmbH (2007) Automotive Semiconductors and Sensors [online], available: <http://www.semiconductors.bosch.de/en/20/flexray/flexray.asp> [accessed 2 October 2007].

Chapter 5 . Performance Analysis

5.1 Introduction

Analysis of software can improve the quality of a system leading to greater satisfaction from the user and ultimately to increased profit for the developers. By analysing a system throughout the software development stage programming errors can be found. This helps to identify errors at an early stage and reduces development time and costs.

Likewise hardware systems need to be analysed. By performing various tests on a system it is possible to identify bottlenecks or shortcomings of a system. An example could be a real-time system that needs to transmit a message over a network before a given time. If the software transmits the message before the given time, the message may still be held up by the driver of the communication device. Likewise the communication hardware may be slow and the message transmission deadline may be missed. The analysis of systems should identify any type of shortcomings in a system. Possible solutions to the problems can then be made based on these observations.

There have been a number of experiments carried out that involve the simulation of networking systems. This chapter will introduce system analysis methods and metrics that have already been implemented. It will also outline some research and techniques that have carried out the simulation of these communication networks.

5.2 System Performance and Analysis

Analysing the performance of real-time systems is an important task. In a real-time system it is essential that deadlines are not missed. By applying performance analysis it is possible to optimise the system. This can reduce or eliminate the chance of a missed deadline.

LITERARY REVIEW

System performance can be classed as response time, Worst-Case-Execution-Time (WCET) and memory-loading. Response time is the time taken between the initialisation of a task and its completion. WCET is the longest time that a computer takes to processes information. Memory-loading is the percentage of available memory to the amount being used (Laplante 1992, p199).

For a FlexRay based system these can be seen as the hardware and software delays. The amount of time it takes a message to pass from a task through the communications controller and onto the communication bus could seriously affect the performance of the system. Other aspects are the processing time for the tasks and the communications schedule. The resource utilisation, such as the amount of memory used, could also affect the performance of the system. Too much memory makes the system costs unnecessary high. Too little allocated memory and messages could be lost or miss deadlines.

The performance and analysis of a system will depend on the nature of the system. If the system is event-triggered there will be a set of measurements and techniques to analyse the system. If a similar system is implemented as a time-triggered system the techniques and measurements could differ. The different performance analysis techniques are outside the scope of this research.

5.2.1 Response Time

The response time of a system will depend on the implementation of the system. Different implementations will lead to different sources of response time delay. The different sources of delay will determine what actions can be applied to reduce the delay (Laplante 1992, p199). The following are examples of possible response time delays.

For polled loop systems there are three different sources of delay: the hardware delay in setting the event, the time to test the event and the time needed to process associated events. The time it takes to process the event and to enter the handler routine can be significant, while the time it takes to process the handler routine will depend on the implementation. This can be made worse by events piling up on each other (Laplante 1992, pp199-200). If there are 'n' overlapping event the response time can be calculated as follows (Laplante 1992, p200):

$$nfP$$

LITERARY REVIEW

where, f is the time needed to check the event and P is the time to process the event.

For an interrupt system there are a number of factors that must be taken into account. Figure 5.1 (Laplante 1992, p201) shows the response time of an interrupt-driven system.

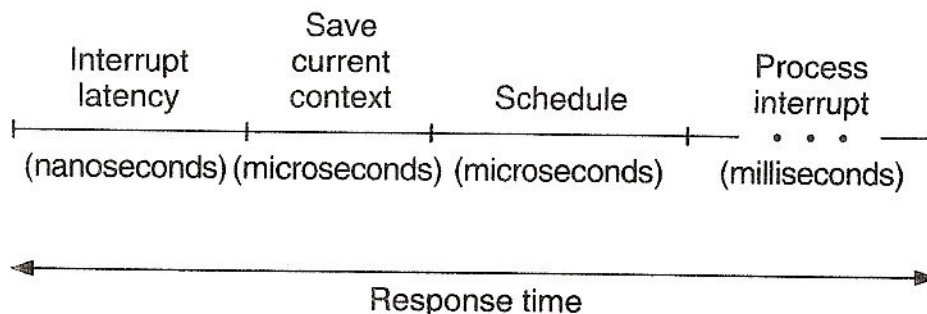


Figure 5.1: Interrupt-driven response time

Interrupt system response time is affected by factors such as the time it takes a system to detect an interrupt and context switch time. The context switching time is the time it takes to transition from the normal program flow to the interrupt handler. The context switch time can be treated as standard executable code when calculating this time. In general the response time for task 'i' (R_i) is given as (Laplante 1992, p200):

$$R_i = L_i + C_s + S_i + A_i$$

where, L is the interrupt latency, C_s is the context save time, S is the scheduling time and A is the execution time.

For a CPU with an interrupt controller and multiple interrupts the schedule time is negligible. When a single interrupt is used with an interrupt controller the schedule time can be calculated by using instruction counting (Laplante 1992, p201). Calculation of the latency can prove difficult however if a lower priority routine tries to interrupt a higher priority routine. The best response time is the time it takes the higher priority task to complete its routine. The worst case cannot be determined as the lower priority may be interrupted itself.

When a higher priority task interrupts a lower priority task the worst case response time is calculated as follows; The propagation delay to signal an interrupt and the CPU detecting this signal (L_p) and the maximum of either the completion time of the longest instruction (L_i) or the maximum time a lower priority task may disable

LITERARY REVIEW

tasks (L_D) (Laplante 1992, p202). This can be expressed as the following formula (Laplante 1992, p202):

$$L_i = L_p + \max\{L_i, L_D\}$$

5.2.2 Worst Case Execution Time (WCET)

The ability to know execution times of modules before the system implementation is important. This can help the system to meet its goals and can even help in the selection of hardware. During the testing it will then be possible to identify the problem modules (Laplante 1992, pp204-5).

To predict or measure the WCET several methods have been developed. These include (Laplante 1992, pp205-210):

Logic Analysers: This is one of the best ways to analyse execution time of a module. It will usually take into account CPU utilisation and hardware latencies. However the software usually needs to be complete.

Instruction Counting: If the software is not complete or a logic analyser is not available this can be employed. It involves tracing the longest path through the code and adding the (maximum) execution times of each instruction.

Pictorial Representations: By employing a bar chart with different shading or colouring a pictorial representation for periodic systems can be achieved. The width of the boxes represents the execution times while the height corresponds to different priority levels. Figure 5.2 (Laplante 1992, p209) shows an example of a timing chart. To construct this chart an interrupt must happen at the appropriate point. If a higher priority task interrupts a lower priority task it can be placed on the graph at that time. The lower priority task will then complete after the higher priority task execution time. If a lower priority task interrupts a higher priority task it is placed after the higher priority task end time. If this is done then an accurate representation is achieved. If the chart cannot be completed then the system is time-over loaded.

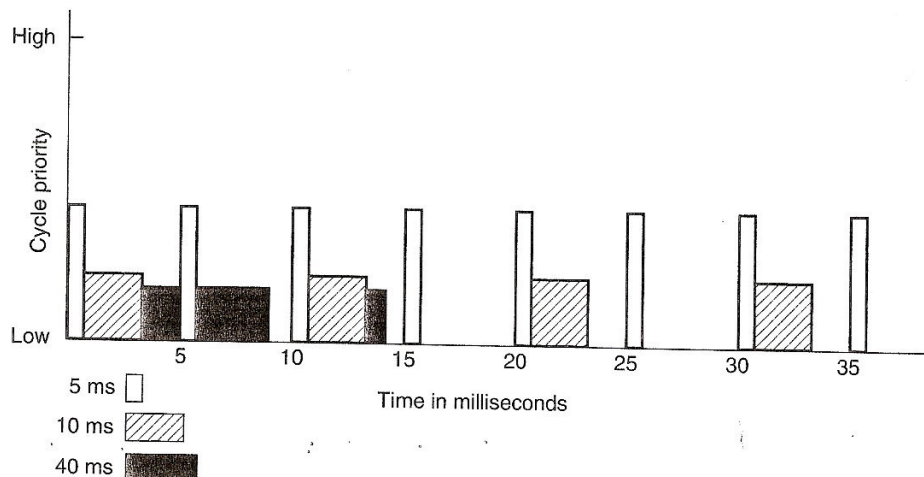


Figure 5.2: Timing chart example

Calculations of the instruction times can require additional information other than that provided by manufacturers. This is due to accesses to I/O devices or memory. To achieve a more accurate time-execution analysis a simulation of the system can be run. This can be configured with various parameters and tests run. However some simulations become very difficult due to complexities in the systems. This is especially true when pipelined systems or RISC architectures are modelled.

5.2.3 Memory-Loading

With memory becoming cheaper and denser the analysis of memory-loading is seen as less of a concern. However where multiple ECUs are present, like in a car where there is a large distributed system, efficient memory use could lead to a large saving (Laplante 1992, p224). For instance in a FlexRay based system there is memory associated with the application and with the communications controller. Therefore efficient use of memory in a FlexRay based system could have a huge impact. Figure 5.3 (Laplante 1992, p225) shows an example of a typical memory map.

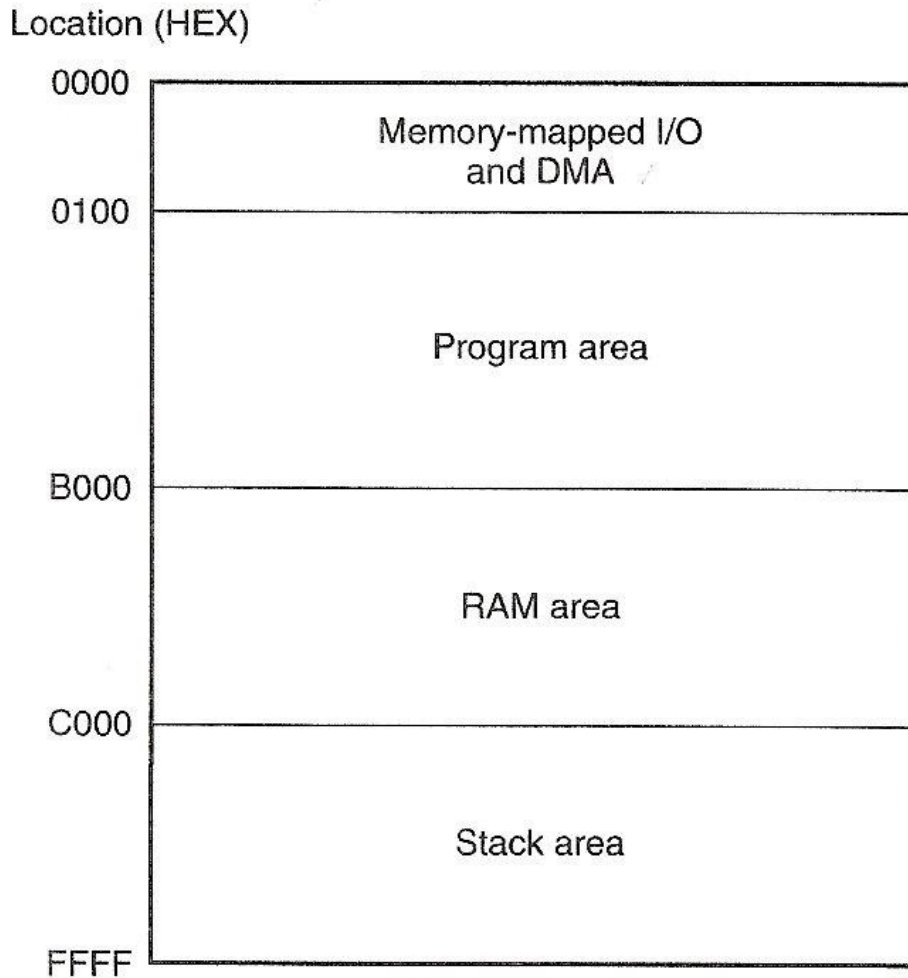


Figure 5.3: Standard memory map

The memory loadings in a system are usually a sum of all the areas in memory. This can be summed up in the following formula (Laplante 1992, p224):

$$M_T = M_P \cdot P_P + M_R \cdot P_R + M_S \cdot P_S$$

where M_T is the total memory-loading, M_P , M_R and M_S are the memory loading for the program, RAM and stack areas respectively. The P_P , P_R and P_S are the percentages of total memory allocated to the different areas.

As the program area is usually stored in ROM it may be treated like RAM for calculations. This is because the RAM size is usually fixed at design time. Therefore to calculate the memory-loading (M) for either area the following formula can be used:

$$M = \frac{U}{T}$$

where U is the number of used locations in memory and T is the total memory area for either the RAM or ROM.

For the stack area the same basic formula can be used. However U is calculated in a different manner. For any task, s , the amount of locations that it requires to store the register, program counter and variables will be defined as C_s . The maximum number of tasks that can be stored in the stack at any stage will be defined as t_{max} . This leads to a formula of U as follows:

$$U_s = C_s \cdot t_{max}$$

5.2.4 Improving Performance Measures

This section is based on sections 9.4 and 9.6 of Laplante (1992, pp210-224; pp227-230). These sections (of Laplante) focus mostly on optimising code to influence the performance of the system. This is due to the fact that the hardware will in many cases be fixed at an early stage. However hardware optimisation could lead to increased performance metrics. As was stated in section 5.2.1.2 simulation is an important tool in analysis of systems. This is one of the best ways to identify any shortcomings of a hardware system and could ultimately lead to improved hardware components.

The basic approach behind reducing response time and time-loading is the identification of wasteful code. This can be due to compilers generating useless code or by poor programming. For instance, floating point numbers take longer to perform calculations on than integer values. If a programmer chooses a floating point variable when an integer variable can be used, then this will increase the execution time unnecessarily. Also there can be waste generated by certain events. An example of this from Laplante is that of a system that employs a temperature sensor. The temperature takes time to measure as the value must pass through an analogue-to-digital converter (ADC). When this is being carried out the application may have to wait for a flag from the ADC to indicate it has finished the conversion. The system will then need to process this information and take any necessary action. However temperature cannot change drastically in most scenarios. Therefore it may be wasteful to measure temperature for example every 10ms. To ensure response times are kept to a minimum, all factors such as a map look-ups or a 32bit divides should be accounted for and steps taken if necessary to reduce the response time where necessary.

LITERARY REVIEW

Some problems are not due to either poor compile time code generation or poor programming. It is therefore necessary to optimise the code. The methods outlined in this section are orientated toward real-time systems. As was stated floating point arithmetic is slower than integer arithmetic. By using a method called 'scaled arithmetic' a reduction in processing time may be achieved. It involves representing numbers as a two's complement number with the least significant bit (LSB) acting as a scale factor and the most significant bit (MSB) acting as a sign indicator. Operations on the number can then be performed and converted to a floating point number at the last step. An alternative method to calculating values at run time is to contain some operations values in a look-up table. This involves pre-calculating values of an operation such as the value of $\text{Cos}(x)$. If the range of values that x can be is known before run time then a look up table can be created. A drawback is, as more points are included more memory is taken up. Also the precision of the values may suffer when using a lookup table.

To help reduce memory-loading there are a number of defined techniques. These include the selection of variables. If a variable is created it will take up space in one area of memory. If this variable holds an intermediate result the variable may not be necessary. By removing this intermediate result and implementing the calculation in a later stage a memory register may be saved. Another form of memory loading is where unreachable code is generated. For example debugging code is never used at run time. It is therefore necessary to determine any code that will never be executed during run time and ensure it is not included at compile time. Other effects could be memory fragmentation. While not an actual form of memory loading, it can produce effects similar to memory-loading. Therefore if possible this should be avoided. Finally the use of bitfields for Boolean variables instead of a byte (or even a word) is also a technique for saving memory.

5.3 Software Metrics

Measures of performance of a system are also known as 'metrics'. Metrics relate to a system designer how well a system performs then intended tasks. This will also lead to more accurate conclusions being drawn from the systems output. By

developing a set of software metrics, an improvement in productivity, development time and product quality can be observed (Möller and Paulish 1993, p8).

5.3.1 The Need for Metrics

Since the 1970's the development of computer hardware has increased at a rate greater than that of software. Processors can now be found with a number of processing cores. This value ranges from 1 to 8 microprocessor cores such as the processor found in the Sony Playstation 3 (one is disabled however). This has resulted in most bottlenecks being traced back to the software (Shepperd and Ince 1993, p8).

This increase in hardware performance causes an increase in the time needed to develop programs as well as affecting reliability. More powerful computers can potentially run more complex and bigger programs in less time than on slower computers. This increase in the size of computer programs as well as their complexity makes them more difficult to troubleshoot. The complexity therefore affects reliability and this trend leads to a need to identify and eliminate any problems if possible at an early stage. Such problems can be bottlenecks of data being passed through a system or where deadlocks/livelocks may occur. Figure 5.4 (Möller and Paulish 1993, p3) shows how a number of factors, related to a badly written piece of software, could affect a company.

Metrics can also be used as a measure of not only software performance, but also of system performance. This could be in the form of the number of messages that pass through a communications controller. Equally the number of messages (of a given size) per microsecond, that a software driver maybe able to pass between a microprocessor and a communications controller could be measured. In this case a bottle neck could be revealed by creating metrics for a given system.

Figure 5.4 could be changed to reflect the poor performance as an indication of, for example, a loss of transmission in a communication system. The inaccurate estimation side of the diagram would reflect a poor system setup, for instance an unnecessarily long communication cycle in a FlexRay based system. This would lead to an overall poor system and reduced system confidence.

LITERARY REVIEW

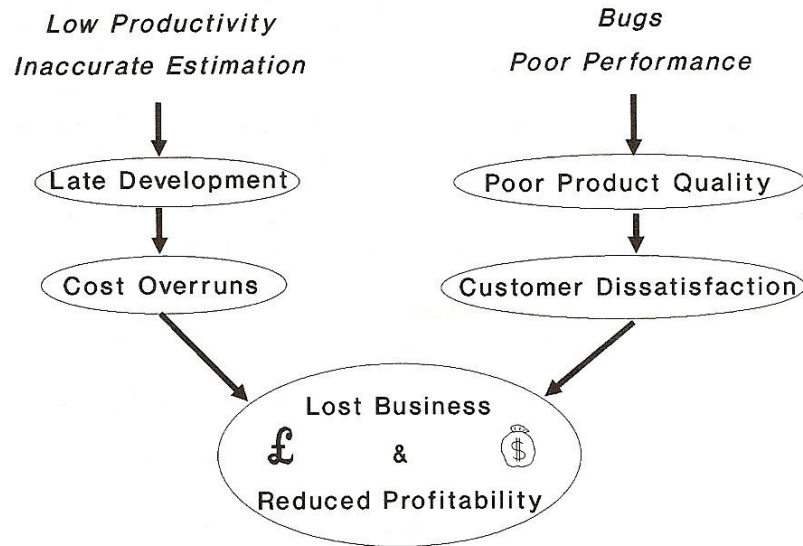


Figure 5.4: The effect of bad software on a company

By employing metrics the following activities can developed to ensure a reduction in cost and increased efficiency in software engineering (Fenton 1991, p9):

- Cost and effort estimation models and measures
- Productivity measures and models
- Quality control and assurance
- Data Collection
- Quality models and measures
- Reliability models
- Performance evaluation and models
- Algorithmic/computational complexity
- Structural and complexity metrics

Figure 5.5 (Möller and Paulish 1993, p71) shows how the use of software metrics can help to find software errors. It is hoped that most errors are found before a system goes to the customer. This helps to highlight how a detailed evaluation of a system can benefit any system. In Figure 5.5, KLOC stands for 'thousand lines of code'.

LITERARY REVIEW

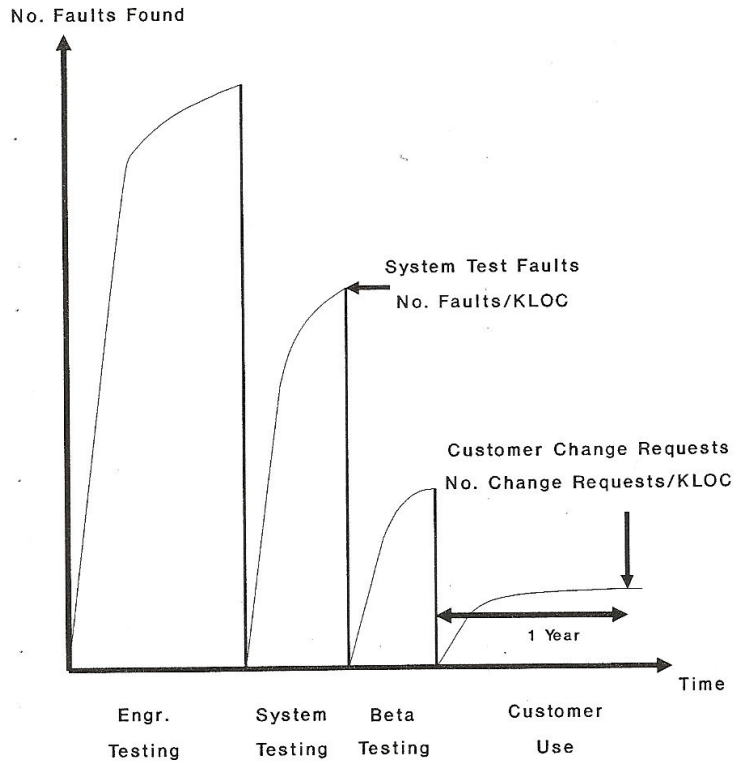


Figure 5.5: Number of faults found in software

By employing metrics a performance measurement will be defined. This will allow stake holders to gain a proper understanding of the different performance aspects. If the metrics have been properly defined they should also eliminate confusion as to what measurements are being defined by the set of performance measures.

5.3.2 System Measurement Framework

This section is based on the framework as described in chapter 3 of 'Software Metrics' (Fenton 1991). It is adapted to be relevant to the research outlined in this thesis.

The various states of any system that are important to identify are any attributes or entities that are of interest to the study. In any system these fall into the following categories:

- Processes
- Products

LITERARY REVIEW

- Resources

Anything that will be of interest in a computer application will usually be related to the above categories. The measurement will be an attribute or entity of one of those categories.

Attributes can be further segmented into internal or external attributes. Internal attributes are processes, products or resources that are related to the system. External attributes on the other hand are how processes, products or resources are related to the system and its environment. These can be seen as reliability or performance attributes. These phrases can be very vague and have many meanings. This makes them more difficult to define and quantify. Table 5.1 (Fenton 1991, p44), shows a selection of possible entities along with examples of both external and internal attributes for each entity example. External and internal attributes may or not affect each other. For example age should not affect the productivity of any member of a workforce, while time to construct a product could possibly impact the cost or quality of the product being developed. The examples shown in Table 5.1 are specific to a software system. However this can be adapted to any generic system.

LITERARY REVIEW

ENTITIES	ATTRIBUTES	
Products	Internal	External
Specifications	size, reuse, modularity, redundancy functionality, syntactic correctness	comprehensability maintainability
Designs	size, reuse, modularity, coupling cohesiveness, functionality	quality, complexity maintainability
Code	size, reuse, modularity, coupling functionality, algorithmic complexity control-flow structuredness....	reliability, usability maintainability
Test Data	size, coverage level,	quality
....
Processes		
Constructing specification	time, effort, number of requirements changes	quality, cost stability
Detailed design	time, effort, number of specification faults found	cost-effectiveness cost
Testing	time, effort, number of bugs found	cost-effectiveness stability, cost
....
Resources		
Personnel	age, price,	productivity, experience, intelligence
Teams	size, communication level, structuredness,....	productivity, quality
Software	price, size,....	usability, reliability
Hardware	price, speed, memory size	reliability,
Offices	size, temperature, light	comfort, quality
....

Table 5.1: Components of software measurement

5.3.2.1 Processes

Processes are system related activities that are normally defined by time. They can be seen as time slices. This may be the time to develop a software function. It could also be the time taken as the software processes a specific task. A metric based on processes can be seen as:

$$\frac{\text{number of messages processed}}{\text{time to run task}}$$

This measure could increase or decrease depending on other messages that are to be processed. It may also be affected by the size of each message.

5.3.2.2 Products

These are taken as deliverables or objects based on the system. This could be in the form of an output a host controller produces based on information it has received from other nodes on a network. The node may then need to send its output over the network. An external attribute that can be applied to this could be the reliability of the system. The time to execute a function, functionality and redundancy are all internal attributes related to this type of product. The internal attributes can be a big factor in relating how good the external attributes are.

5.3.2.3 Resources

Resources are the inputs for the system. These can be number of nodes, type of software driver, available RAM, type of host used. As can be seen these can be individual system components. If there are insufficient resources for a system to perform correctly the level of performance will be affected.

5.3.3 Performance Evaluation and Models

Once the various attributes, entities, resources etc. have been identified it is important to create predictions or assessments based on these measurements. Of the activities listed in section 5.3.1 the 'performance evaluation and models' activity is the most relevant to this research. Again this is taken from Fenton (1991,p13, pp57-58).

The performance evaluation and models activity is generally concerned with the measuring of efficiency. This can be a wide range of metrics, including speed of computation and memory requirements for given inputs. It also covers a wide range of performance metrics corresponding to aspects such as response times and completion rates.

The efficiency attributes are mainly the focus for developers. These attributes are usually concerned with external attributes. For example the type of software driver used in a system could be an attribute. Internal attributes can also be measured even when the machine the application will run on, is not known. This is done by looking at the complexity and type of the system being used. Reasonable analysis can be made by analysing of some internal attributes.

5.4 Previous Systems Analysis

This section will focus on techniques that have already been used to analyse systems. The focus will be on FlexRay performance analysis carried out to date. This section describes the focus of the investigation as well the techniques used to carry out the analysis. It also explores other networking performance analysis techniques including topics such as internet traffic modelling. This should allow the reader to gain an insight into some of the problems faced when analysing a networking system. Included will be sections that look at modelling application software. This presents a significant problem when modelling any system. Without a particular software package to base an application model on, the model may not represent a real world system. If this is the case this could present a significant downfall for any simulation.

5.4.1 FOCUS Modelling of FlexRay

Zhang (2008) introduces the FOCUS modelling language and how it was used to simulate the FlexRay protocol. The FOCUS modelling language is a formal framework for the development of distributed systems. It consists of a range of techniques to formalise specifications with well-defined semantics (Zhang 2008, p334).

The concept model as defined in ‘Modelling and Analyzing of a Time-Triggered Protocol for Automotive Systems’ (Zhang 2008, pp336-339) is as follows:

- Processors are defined as communication controllers or bus guardians. A set is created for each type and these are connected to synchronised clocks. The connections to and from processors are unidirectional. Each processor has a configuration that defines its workings.
- Messages are defined as a set that contain information such as slot, cycle and data.
- An assumption of perfect synchronicity is made. The base time is a slot and it is assumed that the transmission and reception of messages takes no significant amount of time.
- Faults can and do occur. Faults can only occur at processors. If there is a fault then the processor will not produce any result. A component may however produce a subsequent result after a fault. This is because a fault could be the

LITERARY REVIEW

failure of a component to transmit a message during its slot. It may however send a message during another cycle.

Figure 5.6 shows the architectural concept that was used to model the FlexRay system (Zhang 2008, p337).

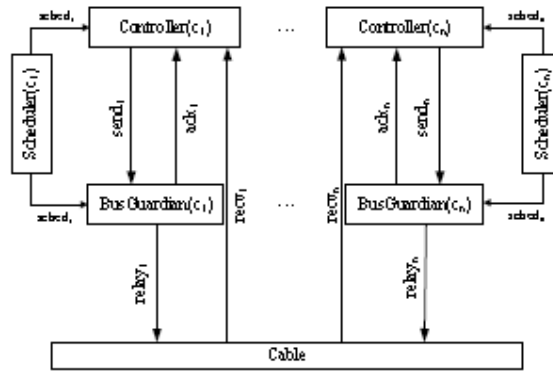


Figure 5.6: FlexRay conceptual architecture

Based on this and the concept model, a formal specification was defined. This methodology and the formalisation allows for the model to be seen as components and their interactions using messages (Zhang 2008, p340).

The FOCUS modelling concept as outlined by Zhang centres on the features used to achieve the model. Zhang (2008) outlines the features of FOCUS as follows:

‘The FOCUS notation uses operators such as union, element of and intersection as the syntax. Functions are also important in FOCUS and operations on streams are mostly defined by functions. The central concept of FOCUS is the idea of streams. Streams are a finite or infinite sequence of elements (also known as messages). The streams can also be defined as timed or untimed streams. The difference between timed streams and untimed streams is that timed streams contain timing information and untimed streams do not contain any timing information. The time is modelled in a discrete manner and assumes a global time divided into intervals known as ticks.’

There is also an emphasis put on modular development. This means that FOCUS sees a component as being made up of a number of related services. To determine the interaction of components a logical expression is used to relate inputs and outputs. The specification for the various components can also be described by

LITERARY REVIEW

using an assumption and guaranteed style, thus splitting the formula into two styles, assumption and guarantee respectively. The assumption are properties that are assumed to be true while guaranteed properties defines the behaviours that always hold if the assumption holds.

The table below, Table 5.2 (Zhang 2008, p337), shows a summary of the type definitions used by Zhang to define the FlexRay protocol.

Type	Definition
Msg	The alphabet of messages.
Msg^ω	Streams over Msg .
$(Msg^*)^\omega$	Timed streams over Msg .
CC	The set of communication controller identifiers.
BG	The set of bus guardian identifiers.
$Slot$	The set of slot identifiers.
$Cycle$	The set of cycle identifiers.
$Config$	The set of node configurations.
Ack	$Ack \triangleq \{ok, fail\}$.

Table 5.2: FOCUS type definitions for FlexRay

Figure 5.7 (Zhang 2008, p339) shows the specification definition for scheduled transmission. This only allows the communication controller for a given slot to transmit. This is just one property needed to have the communication controller operate correctly.

ScheduledTransmission
$send_1, \dots, send_n \in (Msg^*)^\omega; recv_1, \dots, recv_n \in (Msg^*)^\omega;$ $sched_1, \dots, sched_n \in (\{0, 1\}^*)^\omega$
$\forall i \in Slot, p \in CC :$ if $ith(sched_p, i) = \langle 1 \rangle$ then $\exists m \in Msg : ith(send_p, i) = \langle m \rangle$ else $ith(send_p, i) = \langle \rangle$ fi

Figure 5.7: Scheduled transmission definition

It can be seen from Table 5.2 and Figure 5.7 the use of functions and the syntax used in FOCUS.

5.4.1.1 FOCUS Based Modelling, Pros and Cons

The FOCUS modelling approach as was stated is based on a modular development approach. This is a very good approach to development. This can be seen as function development in 'C'. This is a standard practise as it allows for easier testing and debugging of small sections of code. The FOCUS approach also uses mathematical terms to define the operation of modules. These terms such as 'subset of' are standard mathematical terms and many developers would be familiar with it. This would ease the familiarisation stage of learning a new development process.

The mathematical terms could also be seen as a drawback too. The definitions developed using these terms could be difficult to debug/troubleshoot. This could lead to a longer development phase than necessary. The work carried out by Zhang also is small. Larger models may take a lot of specification and computation time. Again if there is a problem with any definition it may be difficult to correct.

5.4.2 FlexRay Based Performance Analysis

Haigescu et. al. (2007) outline a framework for modelling FlexRay based systems. The framework they propose encompasses modelling schedulers and the protocol. They argue that most analysis of FlexRay based systems concentrate on the bus and the scheduling based on this. They also argue that the analysis of the dynamic segment has been overlooked. The dynamic segment is an important section of the FlexRay protocol. It is argued that the dynamic segment if utilised correctly will allow the full advantages of the protocol to be realised.

To test their framework a model of an adaptive cruise control system was developed in Java with a MATLAB front end. The model was based on the diagram as shown in Figure 5.8 with all messages mapped to the dynamic segment (Haigescu et. al. 2007, p289).

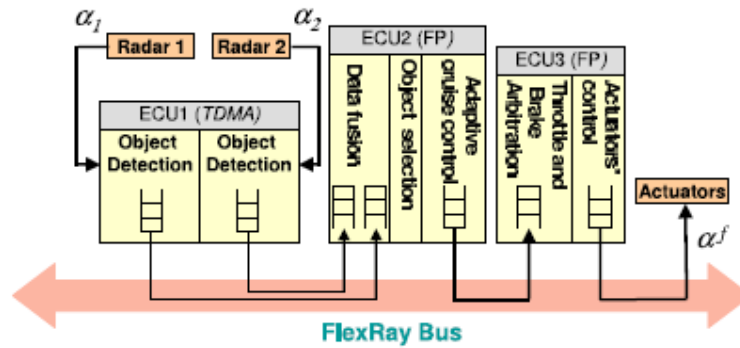


Figure 5.8: FlexRay model system base

The paper also looks at problems and difficulties associated with modelling FlexRay. The focus of the difficulties is on the dynamic segment. This is due to how the dynamic segment works and how it restricts access. For instance the dynamic segment can also be blocked from a node if a higher priority message consumes most or all the dynamic segment. These problems had to be taken into account when developing the framework for their model. Their framework was based on a mathematical framework for analysing the timing properties of multiprocessor embedded systems.

The mathematical models that were used were defined by a number of properties such as task activation rate boundaries and number of activation times. Mathematical models will be looked at in more detail in section 5.4.5. For more information on the specific mathematical models used in the approach outlined above see Haigescu et. al. (2007).

5.4.2.1 Mathematical Modelling, Pros and Cons

Mathematical models attempt to define the system being modelled using mathematical expressions. These expressions can then be analysed to determine the performance of a system. In many cases the mathematical expressions can be easily converted into an executable computer programme using a wide range of software applications. This allows the developer to use the programming language they are most comfortable with or knowledgeable in using. Mathematical models can be found in a wide variety of applications. They are also used in a number of disciplines such as electronics, the sciences and financial areas. Mathematical models and expressions have also been used for a long time.

LITERARY REVIEW

However it can be difficult to accurately define systems using mathematical expressions. The mathematical statement defining the characteristics of a system could affect its precision. As more precision is required the mathematical statement could become very complex. As was stated in section 5.4.1.1 larger models may take a lot of computation time. If there is a problem with any definition it may be difficult to correct. The use of functions can help with this stage. In this case a complex mathematical expression can be broken up into smaller sections and calculated separately before the final result is achieved.

5.4.3 UML Based FlexRay Model

In the paper by Yang et. al. (2005) they propose the use of Unified Modelling Language (UML) when designing system models. The paper presents a development platform that is based on OSEK/VDX. This includes a model design and verification process. This platform can be seen in Figure 5.9 taken from the paper by Yang et. al. (2005, p241).

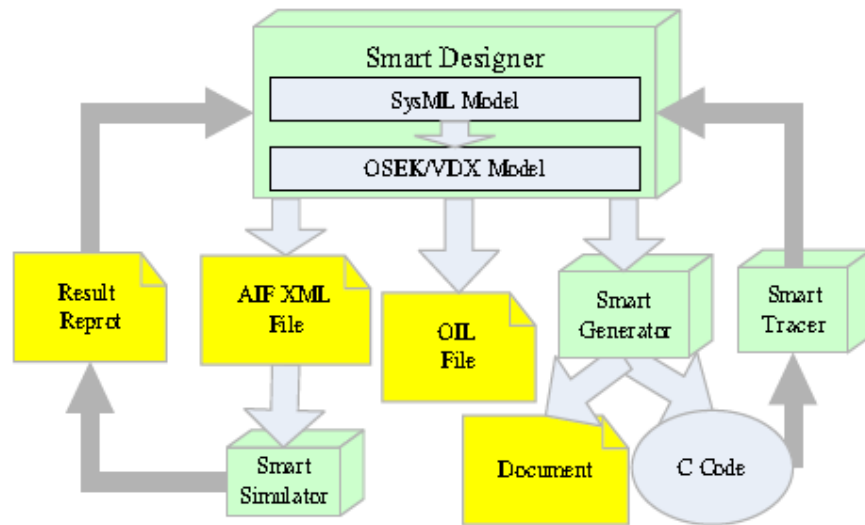


Figure 5.9: System development process.

The approach was achieved by developing SmartOSEK. This is an integrated development environment (IDE). It is split into two modules, one for the OSEK

LITERARY REVIEW

operating system and another to handle the OSEK communication. It provides a graphical design and verification user interaction service.

The system model is split between a framework model and an algorithm model. The framework model describes the complete architecture of the system. The algorithm model describes the implementation of the system algorithms. The algorithm model development can be supported by vendor tools such as Ptolemy and Simulink. However the systems model is described in UML as Smart Designer supports UML.

The UML system model is converted into an OSEK/VDX Model using Smart Designer. Figure 5.10, shows the workflow of the Smart designer (Yang et. al. 2005, p242). The workflow begins with the model editor which allows developers to design the UML model. This model is saved into an .XML file format and the model convertor analyses this file. The model converter then converts the UML model into an OSEK/VDX model by consulting the Model Database. The model database holds information such as objects, relationships and constraints to achieve this. The results converter then passes the processing results onto the Model editor. This is so that the UML model can be modified based on these results. The OSEK/VDX model can then be passed to the smart generator that creates the implementation code for the application.

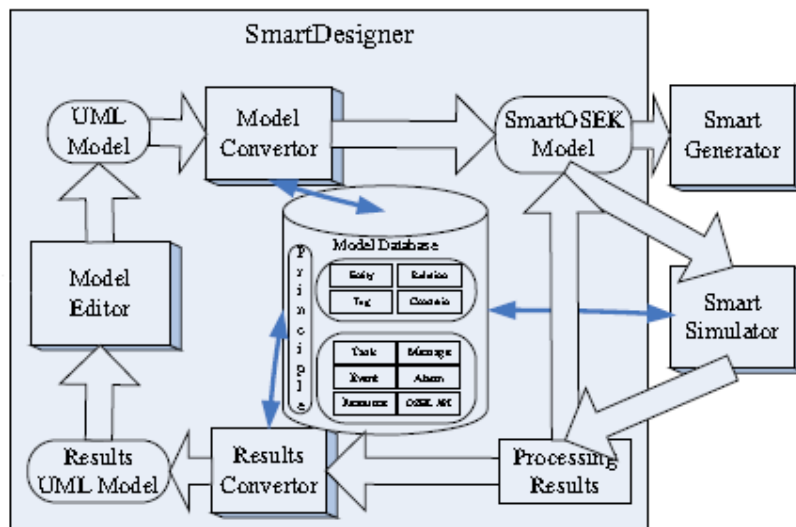


Figure 5.10: The Smart Designer workflow

LITERARY REVIEW

When the OSEK/VDX model is complete it can be verified using the Smart Simulator. The results of this verification process can then be used to modify the system model. To do this the Smart Simulator provides a SmartOSEK COM and SmartOSEK OS simulator to accurately simulate the communication and OSEK scheduling that is compliant with the OSEK/VDX specifications. Figure 5.11 below shows the Smart Simulator system components (Yang et. al. 2005, p241). Note that there is a CAN and J1939 simulator to simulate in-vehicle network communication systems. There is also an interrupt simulator and actuator simulator.

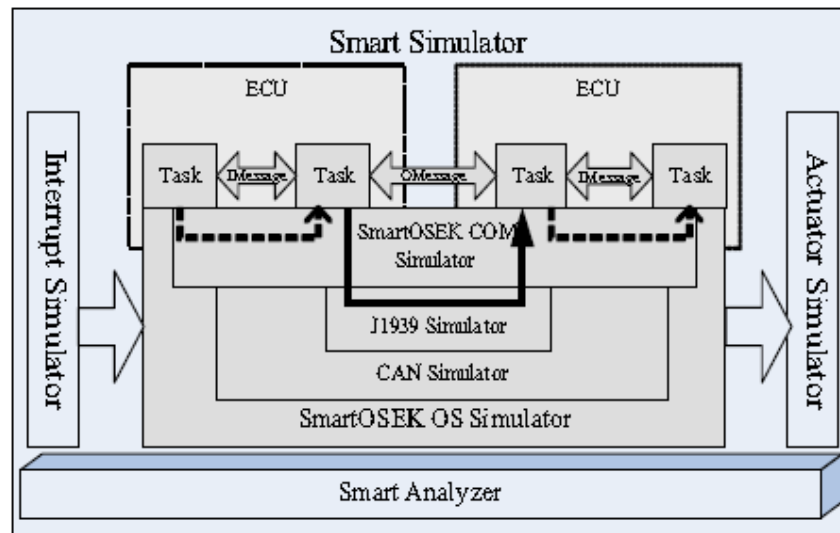


Figure 5.11: The Smart Simulator architecture

Smart Simulator uses Smart Analyzer to provide timing analysis of the model. Built into the Smart Simulator it can deal with mixed pre-emptive and group-based pre-emptive scheduling models.

An example of a model transformed in SmartOSEK can be seen below in Figure 5.12 (Yang et. al. 2005, p244). Using the results of the smart simulator developers can modify the two model types (UML and OSEK/VDX) to refine the system.

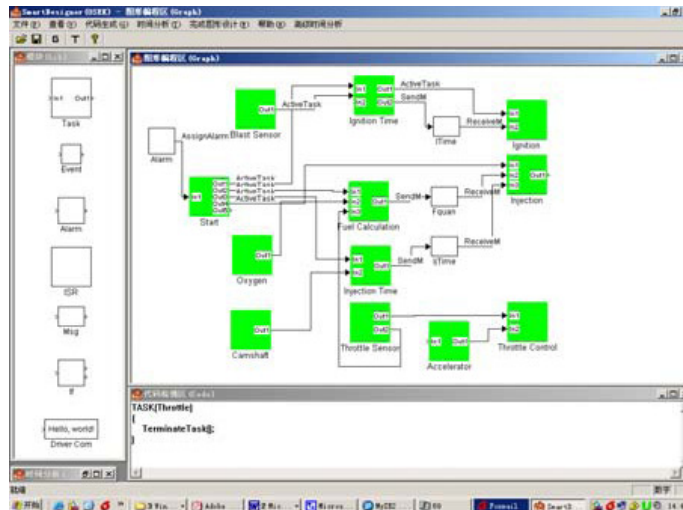


Figure 5.12: SmartOSEK engine control system

5.4.3.1 Smart Designer, Pros and Cons

The workflow again is broken down into separate segments in this method. It also uses a common programming language (UML) to define the models. These models are converted into an OSEK/VDX model that can be analysed to highlight improvements. The program can also produce a set of C code files that can be used in a real world system. This can be a big benefit as a verified application layer can quickly be developed. The designer is restricted to OSEK/VDX models and this means that a limited number of systems can be analysed. There are also a small number of communications protocols that it can simulate. For protocols such as LIN, FlexRay or MOST another simulator would be needed. This may be overcome by modifying the simulator. The Smart Generator stage, while it produces a C file, may not always find the best solution. This code may then require a programmer to optimise the code. This may not always be the quickest solution.

5.4.4 SymTA/S

The papers by Heina et. al. (2005), Richter and Ernst (2006) and Racu et. al. (2007) present a timing analysis technique for automotive and other inter-ECU communications. To accomplish this they propose the use of SymTA/S, a system-level performance and analysis tool developed by Symtvision. The timing analysis

approach is based on formal scheduling analysis techniques and symbolic simulation (Heina et. al. 2005).

The approach taken by SymTA/S is to view components of a system as entities that interact/communicate through the use of event streams. This leads to a well structured model with respect to architecture. It also means that the output stream of one entity is the input stream of another entity. The analysis can then be viewed as a flow of event streams.

Local scheduling analysis algorithms are coupled using event streams. These are described as event models with parameters. Heina et.al (2005) describes an event model with periodic parameter 'P' and jitter, 'J'. They give an example of an event occurring with periodicity of 4 and a jitter of 1. Figure 5.13 (Heina et. al. 2005) shows an 'event stream' that stems from this definition.

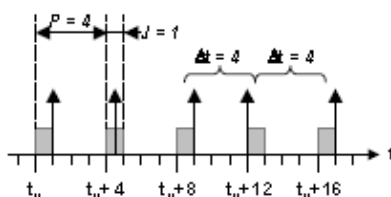


Figure 5.13: Event stream with P=4 and J=1

The gray boxes represent the time where an event may occur.

Events can be used to activate tasks. The activating event can be generated in a number of ways such as time based and external or internal signalling. Each task has an input FIFO and can write to the input FIFO of dependant tasks. In order for the task to execute it needs to be mapped to a communication resource. A scheduler is used to resolve any conflicts with a shared communication resource. Using this worst-case or best-case time analysis can be performed (Heina et. al. 2005). In Figure 5.14 (Heina et. al. 2005) there is a system modelled using SymTAS. In Figure 5.14 the system is a set top box. It receives video signal from rf_video and then sends it to a T.V. by means of a decoder (decryption). Internet traffic (rf_IP) can also be received and sent to a hard drive (hd).

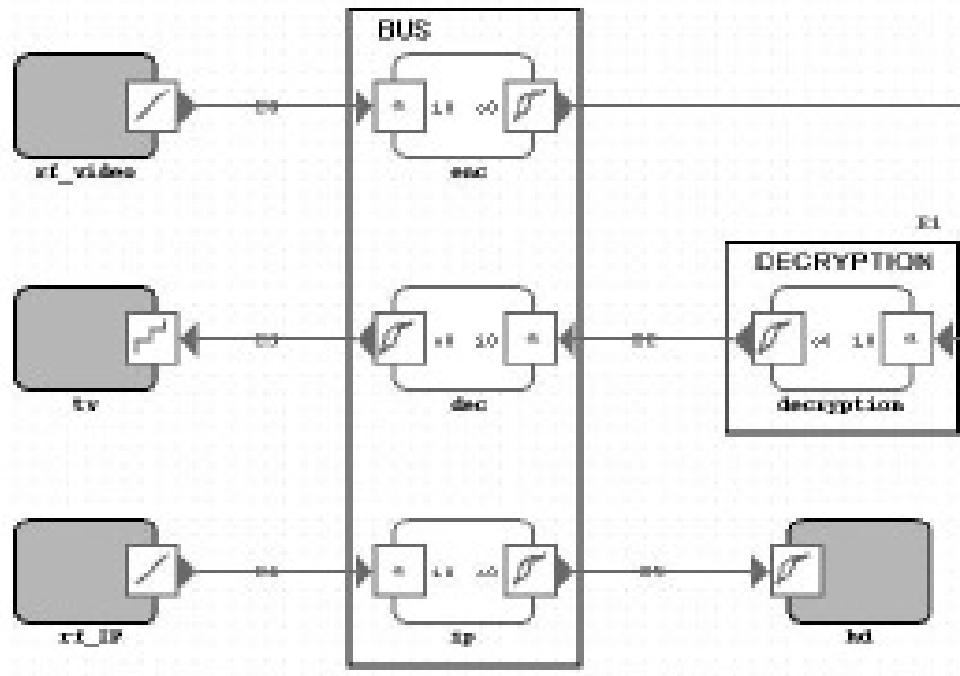


Figure 5.14: SymTAS developed model

Using these models information can be extracted from the models of a given schedule and automatic adaption of the event streams can be done to meet specific demands.

5.4.4.1 SymTAS, Pros and Cons

The benefits of the SymTAS program are very similar to those of discrete event analysis. These benefits are covered in more detail in section 5.4.6.

5.4.5 Mathematical Models

The work outlined in Pop et.al. (2003), Pop et.al. (2006) Pop et.al. (2007) and Pop (2007) concerns the analysis and optimisation of distributed embedded systems such as FlexRay. To achieve this, mathematical models that represent the systems under study were created. Tests were then run and conclusions were drawn from the results.

The paper presented by Pop et. al. (2006) looks at timing analysis of the FlexRay communication protocol. This paper focuses on the analysis of the schedule of a

LITERARY REVIEW

FlexRay node. To do this they developed an application model. The following definitions are some of those as presented in Pop et. al. (2006) and define the application model:

- ‘A’ is a set of acyclic, directed and polar graphs – $G_i(V_i, E_i) \in A$
- A node $\tau_{ij} \in V_i$ is the j-th task/message in G_i
- $e_{ijk} \in E_i$ is an edge from τ_{ij} to τ_{ik} and indicates τ_{ij} is an output that is also the input of τ_{ik}
- A task is ready when all its inputs have arrived and will issue its output after it terminates
- A message is ready after its sender task finishes and is available after its transmission has ended
- Messages passed over a bus are modelled as communication tasks that are inserted on the arc connecting the sender and receiver
- The policy of the scheduling of the tasks is known and the type of transmission is also known (static or dynamic)
- A task $\tau_{ij} \in V_i$ is assigned to execute on Node τ_{ij}
- Task τ_{ij} has a worst case execution time $C_{\tau_{ij}}$
- Communication time of a message ‘m’ is given by $C_m = \text{Frame_size}(m)/\text{bus_Speed}$

The tasks and messages must then be scheduled. This is different for different types of messages and tasks. For instance static messages can be defined in schedule tables while for dynamic messages the worst case execution times must be known first. Once the interactions between the various elements of a system are known, a computer program can be implemented to carry out the analysis of a system. Figure 5.15 below shows a scheduling algorithm (Pop et. al. 2006).

```

GlobalSchedulingAlgorithm()
1  while TT_ready_list is not empty
2  select  $\tau_{ij}$  from TT_ready_list
3  if  $\tau_{ij}$  is a SCS task then
4  schedule_TT_task( $\tau_{ij}$ , Node $_{\tau_{ij}}$ )
5  else //  $\tau_{ij}$  is a ST message
6  schedule_ST_msg( $\tau_{ij}$ , Node $_{\tau_{ij}}$ )
7  end if
8  update TT_ready_list
9  end while
end StaticScheduling

schedule_TT_task( $\tau_{ij}$ , Node $_{\tau_{ij}}$ )
10 find first available time  $t$  moment after  $ASAP_{\tau_{ij}}$  on Node $_{\tau_{ij}}$ 
11 schedule  $\tau_{ij}$  after  $t$  on Node $_{\tau_{ij}}$  so that holistic analysis produces
    minimal worst-case response times for FPS tasks and DYN messages
12 update  $ASAP$  for all  $\tau_{ij}$  successors
end schedule_TT_task

schedule_ST_msg( $\tau_{ij}$ , Node $_{\tau_{ij}}$ )
13 find first ST slot (Node $_{\tau_{ij}}$ ) available after  $ASAP_{\tau_{ij}}$ 
14 schedule  $\tau_{ij}$  in that ST slot
15 update  $ASAP$  for all  $\tau_{ij}$  successors
end schedule_ST_msg

```

Figure 5.15: Scheduling Algorithm

LITERARY REVIEW

There are many elements that define the FlexRay protocol. These are covered in some detail in Pop et. al. (2006). Other papers that can fall into this area are the papers by Kandasamy and Aloul (2005) and Brill et. al. (2006). These papers look at allocating and scheduling messages on a TDMA network and CAN networks respectively. To do this mathematical constraints and relationships are defined for the system under investigation. The system can then be analysed and scheduling of the system can be achieved. The pros and cons of mathematical modelling are covered in section 5.4.2.1.

5.4.6 Discrete Event Simulation

Zhu (2007) and Zhu and Jackman (2007) present a discrete event simulation implementation of an automotive system. Discrete event simulation focuses on the flow of entities around a system. These entities are routed based on attributes and operated on at servers. In this way the flow of information through a system may be modelled. This takes into account the delays in the system and can help find bottlenecks in a system. Chapter 7 of this thesis covers theory related to discrete event simulation in more detail.

The simulation model presented by Zhu and Jackman (2007) was based on a gateway between a CAN network and a FlexRay network. The simulation was designed to accurately model a gateway that met the AUTOSAR specification. To build the model Simulink and SimEvents were used. Figure 5.16, shows the implementation of the upper layer, multicast non TP-PDU transmit model.

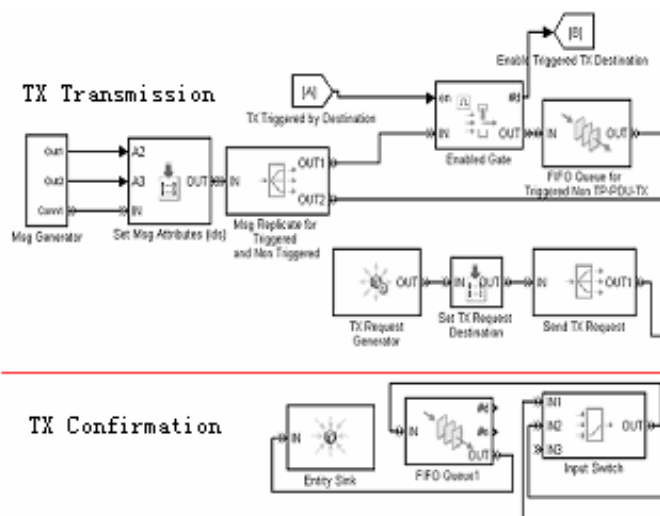


Figure 5.16: Network gateway Simulink/SimEvents model

The use of Simulink/SimEvents allows for modular model building with distinct subsystems and sections of the system. It also allows for a clear flow of entities through a model. This can be seen from Figure 5.16 with the transmission and confirm functions being split into different sections of the model. The different types of simulation software are covered in section 7.8 of this thesis with section 7.9 focusing on MATLAB and Simulink/SimEvents.

5.4.6.1 Discrete Event Simulation, Pros and Cons

Discrete event simulation, as its name suggests, focuses on modelling a system at discrete events in time. It is not concerned with continuous systems. This makes the use of discrete event simulation very suitable for modelling a protocol such as FlexRay or TTP. In these systems all communication happens at discrete points in time. This even happens during the ‘dynamic’ segment of the FlexRay communication cycle. There are also a large number of books or other reference material that covers the theory.

There are drawbacks to discrete event simulation however. One such drawback is that most systems to be modelled are not wholly discrete systems. Many systems have continuous and discrete attributes. This means that the developer must take this into account and make a combined discrete-continuous system. An alternative is to model any continuous elements as discrete elements.

5.4.7 Automesh

Automesh was presented in a paper by Vutturu et.al. (2006). Automesh is a combination of several software model simulators. The paper, by Vutturu et.al. (2006), focuses on the features of Automesh that allow it to be used to carryout performance evaluation of vehicular communications. In particular it looks at a broadcast network scheme that transmits information between vehicles in order to share data such as traffic information. The system under investigation took into account a number of factors such as driver behaviour and geographical topography along with the communication network protocol.

The Automesh architecture therefore takes the form shown in Figure 5.17 (Vutturu et.al. 2006). The five main modules of the Automesh system are:

- The Driving Simulator
- Network Simulator
- Propagation Simulator
- Geographic Database
- Graphical User Interface module

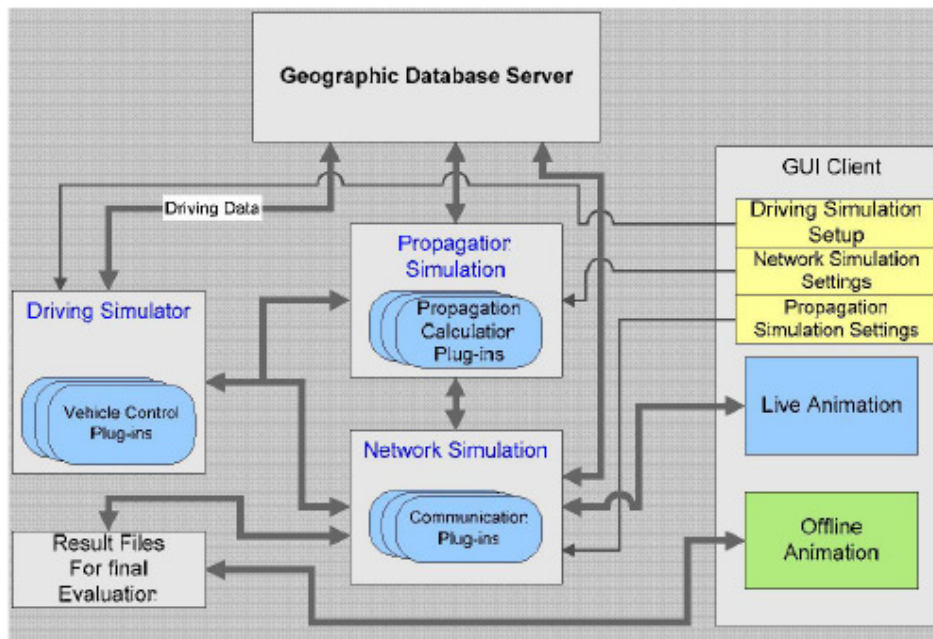


Figure 5.17: Automesh architecture

The driving simulator defines the location of vehicles in the system. The location of the vehicles changes based on a number of factors including information

LITERARY REVIEW

obtained from other vehicles. It also takes into account environmental and vehicle dynamics. These include speed limits and traffic light locations as well as the acceleration characteristics of the vehicles.

The network simulator allows the prebuilt models of communication protocols to be used. The propagation simulator takes into account various factors that could effect the transmission of information between vehicles. This can be affected by factors such as vehicle positioning within a group of buildings among other things. This can be quite complex and as such there is an option to use simple or complex propagation models. As the propagation of signals can be affected by geographical factors there is a need to have a geographical database. This will hold information such as road layout and building information. This can mean a realistic system can be achieved.

The final part of the system is the Graphical User Interface (GUI). This allows for easy configuration of the simulation scenarios as well as providing an animation (in real-time or offline) of the simulation events.

5.4.7.1 Automesh, Pros and Cons

The focus of Automesh is on the communication of information between cars in a given area. This information is intended to communicate information such as traffic jams for example. This will allow drivers follow a different less congested route to their destination. The Automesh simulator therefore takes into account geographical considerations into account. This will help analyse the effectiveness of any wireless communications protocol. This can be useful to a designer of these applications. However for the research as presented in this thesis it would be an unsuitable tool.

There are a number of ideas that may be useful for FlexRay based research. For instance, the propagation simulator could be relevant concept that could be used in a FlexRay model. Ideas from the different analysis techniques may be taken and adapted to produce a more suitable FlexRay analysis tool.

5.4.8 Combined Simulator System

The paper presented by Hatnik and Altmann (2004) discuss the use of simulator coupling. This is to allow the combination of different models that are found in different tool boxes of different software packages. This allows for the best models to be combined, producing a better overall simulation model. The focus of the paper is on modelling a distributed system where data is sent from one or more sources over an Ethernet LAN. Figures 5.18 (Hatnik and Altmann 2004) and 5.19 (Hatnik and Altmann 2004) show an abstract view of a distributed system and how it could be mapped onto the co-simulation environment.

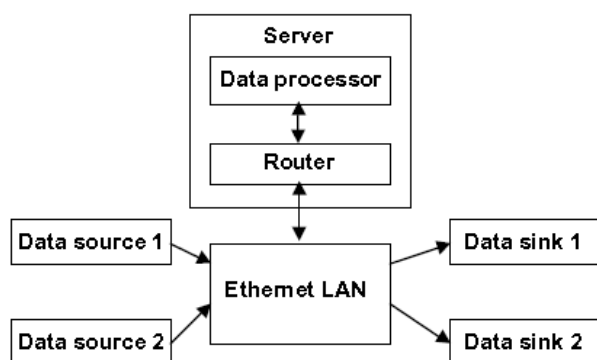


Figure 5.18: Abstract distributed system

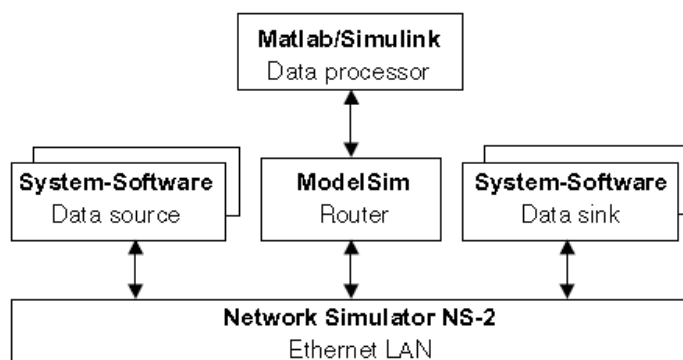


Figure 5.19: Co-simulation mapped example

The data streams are generated and sent to C-applications connected to the Ethernet model. The Ethernet model is constructed in NS-2. The data processor is modelled in Simulink and the router is modelled in ModelSim. Each of these models

must have some way to be connected. As such the communication structure for the whole system is as shown below in Figure 5.20 (Hatnik and Altmann 2004).

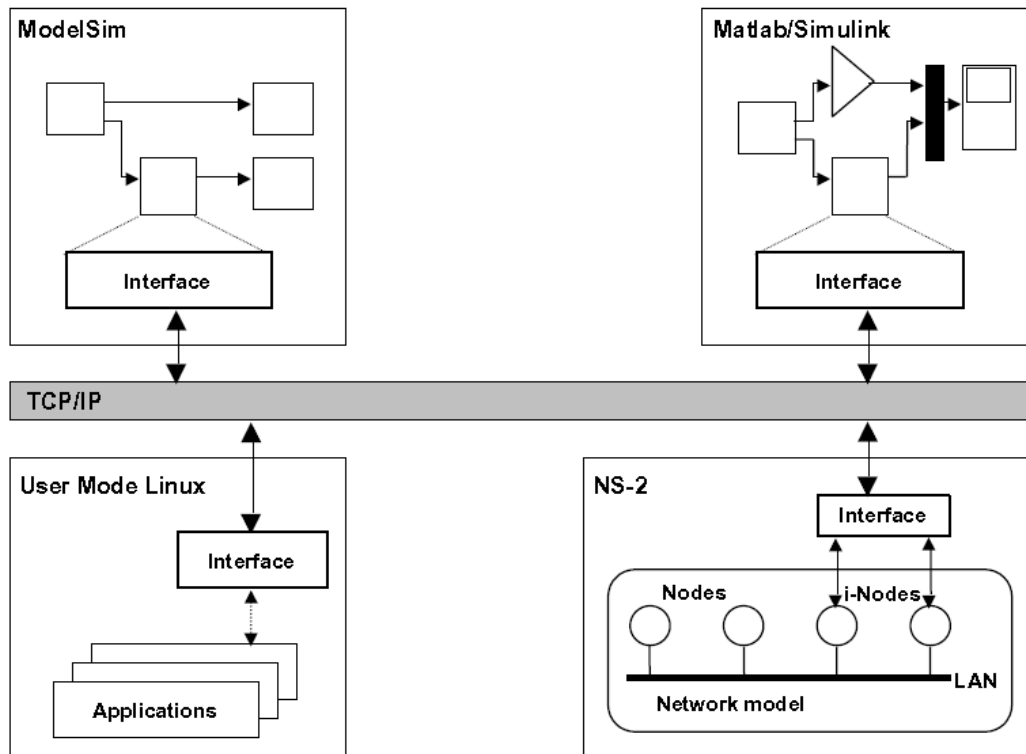


Figure 5.20: Simulator communication structure

As can be seen from Figure 5.20, the models communicate over TCP/IP and use sockets to do so. This allows for the different models to be run on a single computer or on a number of different systems. These systems could be running different operating systems such as Linux, Solaris or Windows. This however means that a coupling component of each model must be implemented and the synchronisation of the simulation has to be done using call backs or blocking read/write socket routines.

NS-2 is a tool to simulate communication network protocols. It also allows for traffic generators to be included to carry out performance and throughput analysis. Abstract client and server models inside the model produce basic loads. These can create or consume packages. The models are described by a set of parameters such as packet size and distribution.

The network model usually consists of a number of node models that contain the necessary node information such as the network stack information. To achieve a

co-simulator however some nodes needed to be modified. The idea of Hatnik and Altmann (2004) was to create interface nodes (i-nodes) to connect to simulator interface. They were designed to allow NS-2 to act as the master simulator. Figure 5.21 below (Hatnik and Altmann 2004) shows their NS-2 model.

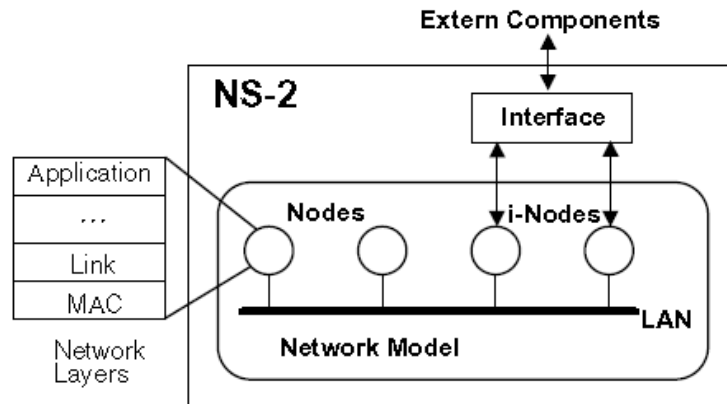


Figure 5.21: NS-2 model showing the interface module

The ModelSim implementation of the router block uses VHDL. It is also possible to use Verilog. VHDL and Verilog are briefly introduced in section 6.11 of this thesis.

A VHDL model will usually consist of an interface and dedicated architectures to describe the operation of the system. The architectures are generally described as behavioural and/or structural implementations. Figure 5.22 (Hatnik and Altmann 2004) shows the behavioural description in VHDL. In this description the model of the router takes header information from the data received. Based on information such as source and destination addresses, the model decides what to do with the data packet.

LITERARY REVIEW

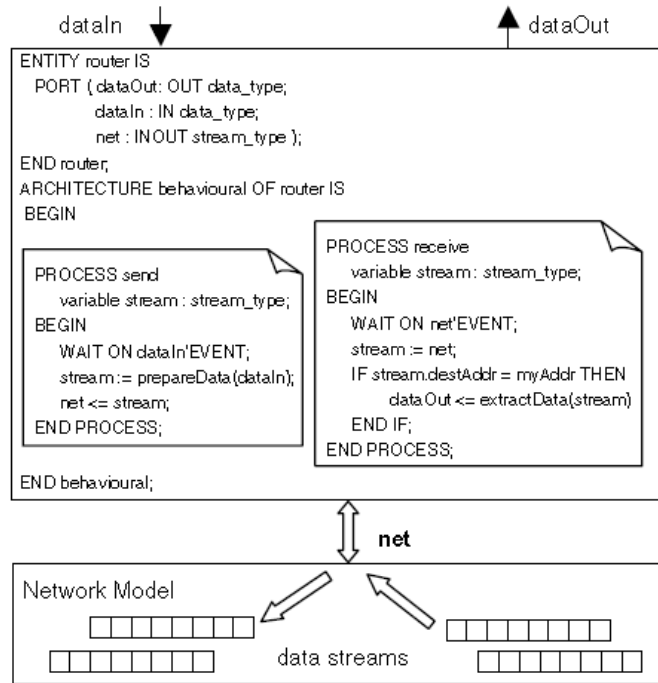


Figure 5.22: Router block model

Simulink was used to implement the server model. As was already stated, section 7.9 of this thesis will cover MATLAB and Simulink in detail. The user mode Linux block shown in Figure 5.20 was implemented as real-world applications to send and receive ‘real data packages’.

5.4.8.1 Combined Simulator Approach, Pros and Cons

The combined simulator approach has many benefits for model developers. The main advantage is the ability to choose the best simulation tool for individual sections of the model. This will help to optimise each model subsection. It also breaks down the model at an early stage. This can help to create a clear concept of the goals to be achieved. The model could also then be run on different machines. This could reduce the run time of the application by spreading the computation over many different processors. This approach also can use an actual network to transmit data. This shows real-world systems can be used instead of a model representation. This reduces the development time due to the communication medium not being modelled. This can also reduce errors in the model.

LITERARY REVIEW

However the use of more than one computer could be costly if you do not have easy access to multiple computers. It may also restrict the time when computers can be utilised to run the tests. For instance this could mean that the computers can only be used at night when they are normally idle. The development of the model may also be affected by this approach. This method of model development requires the developer or developers to be able to use a wide range of software and modelling techniques. If a number of developers are required to build the model it is clear that the cost to do this could be quite high.

From a development perspective problems could occur when running the model if the simulation clock is not correctly synchronised. Other problems could be found also when developing code to get the different applications to communicate correctly. Finally as different simulation methods use different methods to solve/run the simulation model, difficulty could be found when implementing them together.

5.4.9 Previous Analysis Conclusion

A number of different system analysis techniques have been discussed. Each different technique has its own unique set of pros and cons. To ensure the research outlined in this thesis is carried out correctly it is important to first choose the correct system performance analysis technique. By choosing the most suitable technique to perform the FlexRay system analysis more time could be spent developing the necessary analysis tool and less time developing the necessary methods and tasks to perform the analysis. This will result in an overall better and efficient analysis system and ultimately a more accurate set of results.

Each of the different analysis techniques were compared to each other and to the objectives of the research. These objectives included looking at the movement of data around a node and buffer usage. As data moves around the node various constraints must be met. The analysis tool should allow a system designer to analyse the necessary data to optimise a node.

LITERARY REVIEW

	Timing Analysis	Data Flow Analysis	Flexibility	Reference Material	Dedicated Commercial Software	Notes
DES	Yes	Yes	Yes	Yes	Yes	A similar study has already been conducted using DES. A wide variety of software packages can be used and there is a wide variety of reference material available.
Combined Simulator	Yes	Yes	Yes	Yes – separate for each simulator element	Yes	A developer would need to a good standard of a number of different software tools. Problems could be encountered when interfacing the different simulator types as well as the different simulators.
SymTas	Yes	Yes	Yes	No	Yes	Similar to DES but with only one company offering the software. There is also no specific reference material not offered from the development company.
Mathematical Modelling	Yes	Yes	Yes	Yes	No	All mathematical expressions must relate to a variable such as time. The accuracy of the model may be affected by poor or incorrect mathematical relationships.
Automesh	Yes	Yes	yes	No	No	A number of elements of the Automesh such as the propagation simulator may be useful to the study.
UML	Not as standard	Yes	Yes	Yes	No	UML models must be converted into other software languages before execution. This could increase development time.
Focus Modelling	Yes	Yes	Yes	No	No	This has been used to model a FlexRay node already.

Table 5.3: System analysis technique requirements summary

LITERARY REVIEW

Table 5.3 shows a summary of the system analysis technique review. The previous system analysis techniques that have been looked at all scored highly in this review. This is not surprising as they have all been used in the past to successfully perform their intended purposes. To choose the most suitable analysis technique it was necessary to focus the comparison of the techniques on the tools and support available to achieve a successful outcome to the research. This immediately highlighted the combined simulator and discrete event simulation techniques. The other options were discarded for the most-part based on the limited knowledge and support of the techniques and methods used. For example mathematical models were discarded even though a generic programming language may have been used to achieve the ultimate goal but the accuracy may have been affected by poor mathematical expressions. All required methods and tools to simulate a FlexRay node would also need to be defined formally and this would increase the development time.

Discrete event simulation has a large amount of reference material to help develop a model. The software available has, in some cases, been available for a number of years and is widely understood. This means that a large amount of work can be saved by using the methods and techniques already developed for any specific software that may be chosen. The time spent developing the model can then be dedicated to creating the model rather than learning how to use the modelling software. There are also a large number of online support forums for a number of the different modelling software programs. This is an advantage as this means expert knowledge on the software programs can easily be consulted.

The combined simulator approach allows the best simulation model to be developed. By breaking down a system into various subsystems a clear picture of the operation can be achieved. The most suitable modelling technique and software can then be chosen to model any aspect of the system. This has drawbacks however as it requires a developer to be knowledgeable in a number of different modelling techniques and software packages. The different subsystems must ultimately be combined into an overall system simulation model. This could lead to problems as different simulation techniques will represent different aspects of the system, time for example, in different ways. It could then prove difficult to combine all the different modelling subsystems and could slow the execution of the model down as one subsystem may have to wait for a following subsystem to complete a task.

LITERARY REVIEW

Due to the issues of the combined simulator approach it was decided that discrete event simulation should be used.

5.5 Conclusion

System performance analysis is an important step in product development. It forces the developer to focus on the system under development from an early stage, i.e. the initial development stage, through to the release of the system. However to achieve this, a proper set of metrics must be developed. This will allow the accurate interpretation of the analysis results. By defining a good set of metrics early on in the design stage of a system a better product can be delivered. The metrics can be used to help investigate the performance of the actual system under investigation. They can also be applied to the project as a whole. When the project is complete, system analysis helps to quantify how well the system performs. This includes the execution time and memory requirements. It can also help to judge if the application is suitable for its intended purpose.

As can be seen from section 5.4 a large number of different ways of simulating and analysing systems has been explored. New methodologies have also been suggested based on these methodologies and these have been explored. Each simulation method was then analysed for its suitability to conduct the research presented in this thesis. However there has not emerged a single method that is better than any other method. The method chosen should be suitable to the properties of the system under investigation.

In section 5.4.6 discrete event simulation was introduced. This focuses on the amount of time it takes for an entity to be serviced before moving onto another part of the system. Where the entity goes could depend on attributes associated with the entity. This approach lends itself to performance analysis of data flow through a system. This is a big advantage for the research described in this thesis. For this reason it was chosen as the most suitable analytical approach.

Other factors that were not covered however would be cost, support and availability, etc. of the software to be used. These were all also considered before

LITERARY REVIEW

discrete event simulation was decided upon. The factors that were considered when choosing the most suitable software are covered in chapter 7.

5.6 References

Bril, R.J., Lukkien, J.J., Davis, R.I. and Burns, A. (2006) Message Response Time Analysis for Ideal Controller Area Network (CAN) Refuted, University of Eindhoven, Computer Science Report, May 2006.

Fenton, N.E. (1991) Software Metrics – a Rigorous Approach, London: Chapman & Hall.

Hagiescu, A., Bordoloi, U.B., Chakraborty, S., Sampath, P., Vignesh, P., Ganesan, V. and S. Ramesh, S. (2007) Performance analysis of FlexRay-based ECU networks Annual Proceedings of the 44th annual ACM IEEE Design Automation Conference, San Diego, California, 284 - 289 .

Hatnik, U. and Altmann, S. (2004) Using ModelSim, Matlab/Simulink and NS for Dimulation of Distributed Systems, Proceedings of the International Conference on Parallel Computing in Electrical Engineering, Dresden, Germany, September 7-10 2004, IEEE Computer Society Washington, DC, USA, 114 – 119.

Henia, R., Hamann, A. and Jersak, M. (2005) System Level Performance Analysis- the SymTA/S Approach, IEE Proceedings Computers and Digital Techniques, 152(2), 148-166.

Kandasamy, N. and Aloul, F. (2006) The Synthesis of Dependable Communication Networks for Automotive Systems, SAE 2006 Transactions Journal of Passenger Cars: Electronic and Electrical Systems, SAE International, Warrendale, Pennsylvania, USA.

Laplante, P.A. (1992) Real-Time System Design and Analysis an Engineer's Handbook, New Jersey: IEEE Press.

LITERARY REVIEW

Möller, K.H. and Paulish, D.J. (1993) Software Metrics – a Practitioner’s Guide to Improved Product Development, London: Chapman & Hall.

Pop T. (2007) Analysis and Optimisation of Distributed Embedded Systems with Heterogeneous Scheduling Policies, unpublished thesis (PhD.), Linköpings Universitet.

Pop, P., Eles, P. and Peng, Z. (2003) Schedulability Analysis and Optimisation for the Synthesis of Multi-Cluster Distributed Embedded Systems, IEE Proceedings - Computers & Digital Techniques, Vol. 150, Issue 5, Sept. 2003, pp. 303-312.

Pop, T., Pop, P, Eles, P. and Peng, Z. (2007) Bus Access Optimisation for FlexRay-Based Distributed Embedded Systems, Proceedings of the Conference on Design, Automation and Test in Europe, Nice, France, April 16-20 2007, IEEE Computer Society Washington, DC, 51 – 56.

Pop, T., Pop, P., Eles, P., Peng, Z. and Andrei, A. (2006) Timing Analysis of the FlexRay Communication Protocol, in Proceedings of the 18th Euromicro Conference on Real-Time Systems, Dresden, Germany, July 5-7, 2006, IEEE Computer Society Washington, DC, USA, 203-216.

Racu, R., Ernst, R., Richter, K. and Jersak, M. (2007) A Virtual Platform for Architecture Integration and Optimization in Automotive Communication Networks, Proceedings of SAE World Congress, Detroit, Michigan, USA, April 16-19, 2007.

Richter, K. and Ernst, R. (2006) Real-Time Analysis as a Quality Feature: Automotive Use-Cases and Applications, Proceedings of Embedded World Conference, Nürnberg, Germany, February 14-16, 2006.

Shepperd, M. and Ince, D. (1993) Derivation and Validation of Software Metrics, Oxford: Oxford University Press.

Vuyyuru, R., Oguchi, K., Collier, C. and Koch, E. (2006) Automesh: Flexible Framework for Vehicular Communication ,Proceedings of 2006 Third Annual

LITERARY REVIEW

International Conference on Mobile and Ubiquitous Systems: Networking & Services, San Jose, California, USA, July 17-21 2006, IEEE Computer Society Washington, DC, USA, 1-6.

Yang, G., Zhao, M., Wang, L. and Zhaohui Wu, Z. (2005) Model-based Design and Verification of Automotive Electronics Compliant with OSEK/VDX, Proceedings of the Second International Conference on Embedded Software and Systems (ICESS'05), Xi'an, China, December 16-18 2005, IEEE Computer Society Washington, DC, USA, 237 – 245.

Zhang, B. (2008) Modelling and Analyzing of a Time-Triggered Protocol for Automotive Systems, Proceedings of Third International Conference on Systems, Cancun, Mexico, April 13-18, 2008, IEEE Computer Society Washington, DC, USA, 334-340.

Zhu W. (2007) Performance Analysis of AUTOSAR Vehicle Network Gateways, unpublished thesis (M.Sc.), Waterford Institute of Technology.

Zhu W. and Jackman, B. (2007) Using Simulation for Designing In-Vehicle Network Gateways, SAE 2007 Transactions Journal of Engines, SAE International, Warrendale, Pennsylvania, USA.

Chapter 6 . E-Ray

6.1 Introduction

The Bosch E-Ray communications controller was developed to fully conform to the FlexRay protocol and has been conformance tested successfully (Robert Bosch Gmbh 2006a; FlexRay Consortium 2006). It is a full FlexRay IP-Module with message handling, has driver support from reputable companies and is one of the most widely used FlexRay modules (Robert Bosch Gmbh 2006a). It is available as an FPGA netlist or as VHDL source code. The message RAM holds up to 128 message buffers and each message buffer can hold up to 254 data bytes (FlexRay Consortium 2006), depending on the configuration of the chip for a given application.

All the messaging functions such as message acceptance or rejection and the schedule for messages are handled by a message handler. The registers of the module can also be accessed by an external CPU via a host interface. This can then be used to control or change various aspects of the module. This could include the protocol controllers, interrupt control or access to the message RAM as well as other aspects of the module's features (FlexRay Consortium 2006).

6.2 Features

Some of the features of the E-Ray module are (Robert Bosch Gmbh 2006b; FlexRay Consortium 2006):

1. 100% conformance with the FlexRay protocol specification v2.1.
2. Dual channel with up to 10Mbits/s data rate on each as defined in the FlexRay Consortium protocol (2005).
3. Configurable message RAM.
4. Host access to message buffers.
5. Filtering for frame, channel ID and cycle counter values.

6. Support for network management.
7. Maskable module interrupt.
8. 8/16/32-bit generic interface for connection to customer-specific host CPUs.

6.3 Components

Figure 6.1 shows a block diagram of all the components of an E-Ray module. It is based on the block diagram found in the Bosch product information (2006a).

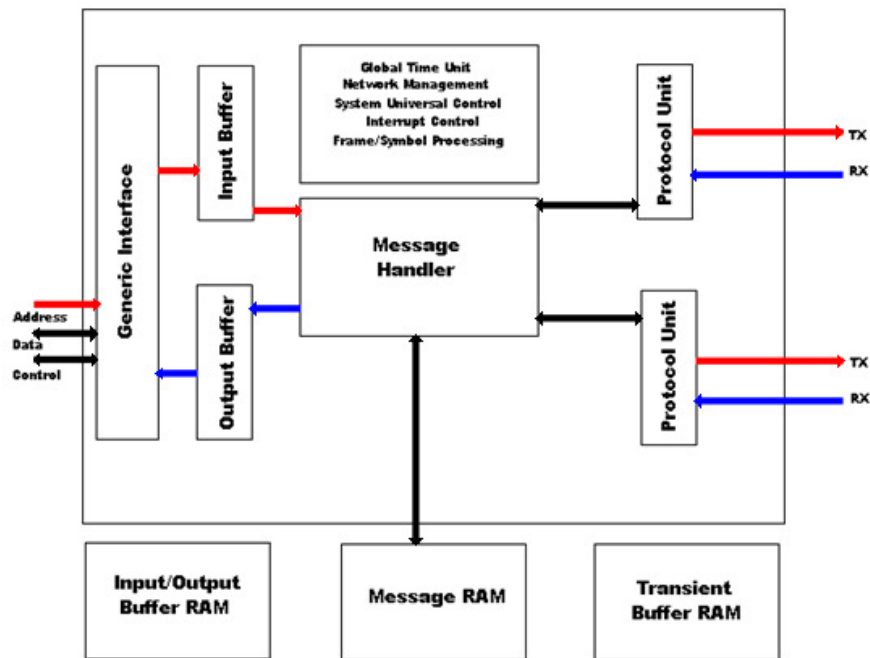


Figure 6.1: Block diagram of the workings of an E-Ray chip

The red lines in Figure 6.1 represent data flowing from left to right; the blue lines are data flowing from right to left. The black connecting lines represent data flow in both directions.

6.3.1 Module Functions

6.3.1.1 Generic Interface

This allows a customer-specific CPU to be connected to the E-Ray IP-module through an 8, 16 or 32-bit generic interface (Robert Bosch GmbH 2006a; Robert Bosch GmbH 2006b).

6.3.1.2 Input and Output Buffers

These allow storage of two complete messages each. This is for transfer between the host and the message RAM. The input/output buffer RAM is 8448 bits in size and is broken down into $4 \times 64 \times 32$ bits (Robert Bosch GmbH 2006a; Robert Bosch GmbH 2006b).

6.3.1.3 Message Handler

This controls the data transfer between the input/output buffers and the message RAM. It also controls the transfer between the protocol units transient buffers and the message RAMs (Robert Bosch GmbH 2006a; Robert Bosch GmbH 2006b).

6.3.1.4 Message RAM

The message RAM is a single-ported RAM which stores the configuration and FlexRay messages. The message RAM is 4,352 bits in size (Robert Bosch GmbH 2006a; Robert Bosch GmbH 2006b).

6.3.1.5 Global Time Unit

This is a common base for both channels and provides the microtick and macrotick, clock synchronisation, cycle counter and the timing control for the static and dynamic segments of the communications cycle (Robert Bosch GmbH 2006a; Robert Bosch GmbH 2006b).

6.3.1.6 Network Management

This handles the network management algorithm (Robert Bosch Gmbh 2006a; Robert Bosch Gmbh 2006b).

6.3.1.7 System Universal Control

This controls wakeup, startup and integration of the node into a cluster (Robert Bosch Gmbh 2006a; Robert Bosch Gmbh 2006b).

6.3.1.8 Interrupt Control

This controls generation of module interrupts. The interrupt flags and timers can be found here (Robert Bosch Gmbh 2006a; Robert Bosch Gmbh 2006b).

6.3.1.9 Frame/Symbol Processing

This is where the timings of frames and symbols are enforced. It also tests the received messages for errors such as corrupted data and syntax and sets the slot status flags (Robert Bosch Gmbh 2006a; Robert Bosch Gmbh 2006b).

6.3.1.10 Protocol Unit

This is the connection to the physical medium of the network. It consists of a shift register and protocol finite state machine (FSM). The transient buffer RAM is connected to these for temporary storage. The transient buffer RAM is 8448 bits in size which is broken down as $2 \times 128 \times 33$ bits (Robert Bosch Gmbh 2006a; Robert Bosch Gmbh 2006b).

6.4 Register Map

The E-Ray module has an address space of 2K bytes. This gives an address range of 0x0000 to 0x07FF. The address space of the E-Ray in the range 0x0000 to 0x000F are reserved for customer specific CPUs. The function of these changes from

LITERARY REVIEW

CPU to CPU. The registers are organised into 32-bit wide registers but have the ability to be accessed as 8 or 16 bit registers (Robert Bosch Gmbh 2006b, p18).

The message buffers are assigned according to the following table, Table 6.1 (Robert Bosch Gmbh 2006b, p 18).

Message Buffer 0	↓ Static Buffers	
Message Buffer 1		
...	↓ Static + Dynamic Buffers	⇐ FDB
	↓ FIFO	⇐ FFB
Message Buffer N-1		
Message Buffer N		⇐ LCB

Table 6.1: Message buffer assignment

The message buffers can be assigned in various ways. To configure the message buffers the Message RAM configuration register should be set as desired. The maximum number of message buffers that can be assigned is 128 and the maximum length of the payload is 254 bytes. The number of buffers however depends on the configured maximum length of the system's payload (Robert Bosch Gmbh 2006b, p18). As can be seen in Figure 6.2, there are three sections that the memory has been divided into:

1. The static buffers
2. The static or dynamic buffers
3. The FIFO

The first section is used for the static section only. The first message buffer, buffer 0, is used to hold the startup/sync frame or the single slot frame if the node can transmit one. If these are different for both channels then buffer 1 is used to store the other channel's sync or single slot frame (Robert Bosch Gmbh 2006b, p18).

The second group is used for buffers assigned to either the static or dynamic segments. They can be reconfigured during runtime to suit the given situation (Robert Bosch Gmbh 2006b, p18).

The buffers belonging to the third and final group can be assigned to a receive FIFO.

6.5 Communication Controller States

This section will describe the different states the controller can be in and what they do. Figure 6.2, shows all possible states and the possible transitions in and out of each state. This diagram is based on two diagrams in the E-Ray's user manual (Robert Bosch GmbH 2006b, p105; Robert Bosch GmbH 2006b, p110).

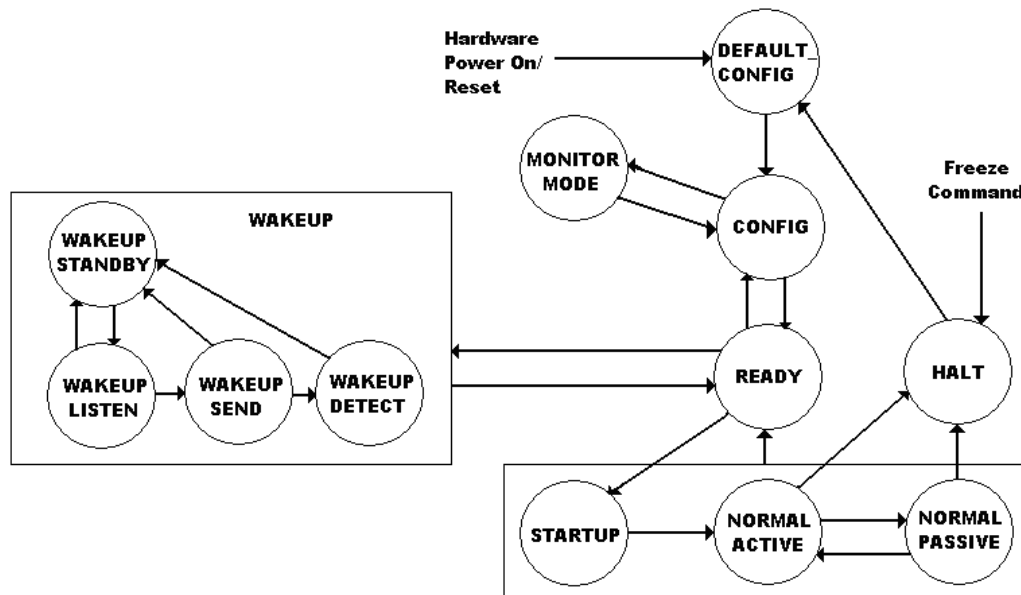


Figure 6.2: Possible communications controller states

Each state and some details on each are given in sections 6.5.1-6.5.9.

6.5.1 DEFAULT_CONFIG

In the DEFAULT_CONFIG state the communications controller is stopped and all registers are accessible. The physical pins connecting the device to the physical layer are also set to an inactive state. As can be seen from the diagram, when leaving this state the controller can only transition to the CONFIG state and to enter this state the controller must be powered on/ reset or transition from the HALT state (Robert Bosch GmbH 2006b, p107).

6.5.2 CONFIG

The controller will enter and remain in this state with the physical pins still inactive. This state is used to configure the controller and if this state has been arrived at through the HALT-DEFAULT_CONFIG route additional information will be available to the host to ensure that the setup is fault free. To exit from this state an unlock sequence is used using the lock register (Robert Bosch GmbH 2006b, p107).

6.5.2.1 Unlock Sequence

The unlock sequence is a three write process. The first write to the lock register is with the value 0xCE. This is followed by a second write with the value 0x31 followed by the third write with the command you wish i.e. READY, MONITOR_MODE, ATM or LOOP_BACK, to the SUC configuration register 1.

If the write sequence is interrupted by other write accesses between the second and third write then the controller stays in the CONFIG state and the process must be repeated (Robert Bosch GmbH 2006b, p23).

6.5.3 MONITOR MODE

This mode is used to receive frames and detect wakeup patterns. This state can be used to debug the system. An example of this is if a startup of a FlexRay network fails. In this mode it may only be possible to receive messages on only one channel (Robert Bosch GmbH 2006b, p108).

6.5.4 READY

This state is used to either transition to the WAKEUP state or STARTUP state. This is so that the node will be able to wakeup a cluster or integrate into a running cluster (Robert Bosch GmbH 2006b, p108).

6.5.5 STARTUP

Any node entering startup will follow the steps as shown in Figure 6.3 (Robert Bosch GmbH 2006b, p114).

LITERARY REVIEW

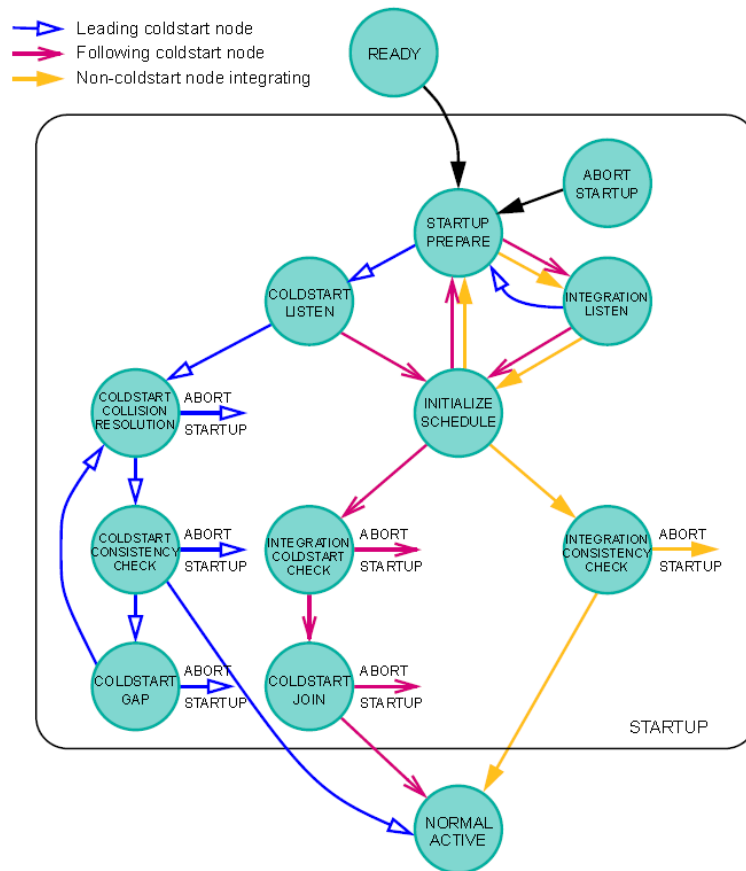


Figure 6.3: State diagram for node entering startup

Figure 6.3 shows the standard integration method of a node. For further details on the startup procedure see the Bosch application note on the startup (Robert Bosch GmbH 2006d), the FlexRay protocol (FlexRay Consortium 2005) and the E-Ray's user manual (Robert Bosch GmbH 2006b, pp113-7).

6.5.6 NORMAL ACTIVE

A node entering the NORMAL_ACTIVE mode will start to communicate across a cluster in the way defined by the FlexRay protocol (FlexRay Consortium 2005). This means that the node sends and receives FlexRay frames and performs synchronisation. The host interface is also operational (Robert Bosch GmbH 2006b, p118).

6.5.7 NORMAL PASSIVE

The NORMAL_PASSIVE state is entered when the error state changes from active to passive. In this mode frames are received over the physical medium but no transmissions occur from the node. The host interface will also be operational and clock synchronisation will still occur (Robert Bosch GmbH 2006b, p118).

6.5.8 HALT

This state is entered when a freeze command is received in any state; a halt state is entered when the controller is in either the NORMAL_ACTIVE or NORMAL_PASSIVE states or when exiting from either the NORMAL_ACTIVE or NORMAL_PASSIVE state because the counter for the clock correction failed limit was reached. The state that the controller was in prior to entering the HALT state is held in the CC Status Vector register (Robert Bosch GmbH 2006b, p108).

6.5.9 WAKEUP

This section describes the operation of the controller's states for wakeup. For a full description of the wakeup procedure of a FlexRay node see the FlexRay protocol (FlexRay Consortium 2005) and the Bosch wakeup application note (Robert Bosch GmbH 2006c).

6.5.9.1 WAKEUP STANDBY

This state allows transition from various other states to the WAKEUP_LISTEN or READY states.

6.5.9.2 WAKEUP LISTEN

In this state the controller listens for wakeup patterns (WUP) sent from other nodes in the cluster. The state is controlled by two timers. One will allow the wakeup of a cluster faster in a non-noisy environment while the other is used where noise-interference is an issue (Robert Bosch GmbH 2006b, p111).

6.5.9.3 WAKEUP SEND

This state transmits a wakeup pattern while checking for collisions on a given channel. After successful wakeup of a cluster the node must enter the startup mode (Robert Bosch Gmbh 2006b, p111).

6.5.9.4 WAKEUP DETECT

This state attempts to indentify the reason a wakeup collision was detected. If it cannot determine another wakeup attempt by another node or ongoing communication on the channel as the reason for a collision within a given time, the node leaves this state and the reason for the collision is set as unknown (Robert Bosch Gmbh 2006b, p111).

6.6 Error Handling

The implemented error handling procedure is intended to allow all non-affected communicating nodes to continue operating as normal if a single node experiences a lower layer error (Robert Bosch Gmbh 2006b, p103). Table 6.2 (Robert Bosch Gmbh 2006b, p103), shows the error modes along with how the controller behaves during this time.

LITERARY REVIEW

Error Mode	Activity
ACTIVE (green)	Full operation , State: NORMAL_ACTIVE The CC is fully synchronized and supports the cluster wide clock synchronization. The host is informed of any error condition(s) or status change by interrupt (if enabled) or by reading the error and status interrupt flags from registers EIR and SIR.
PASSIVE (yellow)	Reduced operation , State: NORMAL_PASSIVE, CC self rescue allowed The CC stops transmitting frames and symbols, but received frames are still processed. Clock synchronization mechanisms are continued based on received frames. No active contribution to the cluster wide clock synchronization. The host is informed of any error condition(s) or status change by interrupt (if enabled) or by reading the error and status interrupt flags from registers EIR and SIR.
COMM_HALT (red)	Operation halted , State: HALT, CC self rescue not allowed The CC stops frame and symbol processing, clock synchronization processing, and the macrotick generation. The host has still access to error and status information by reading the error and status interrupt flags from registers EIR and SIR. The bus drivers are disabled.

Table 6.2: Error modes

Below is a detailed description of how the controller deals with various events.

6.6.1 Freeze Command

If a severe error is detected by the host then it can transition the controller to the HALT state from any other state by using the FREEZE command. The protocol operations control state from which the HALT state was entered can be read from the CC status vector register (Robert Bosch Gmbh 2006b, p104).

6.6.2 Halt Command

The host may from time to time wish to stop the controller by using the HALT command. This will stop communication on the node and if the controller is in NORMAL_ACTIVE or NORMAL_PASSIVE mode this will happen at the end of the current communication cycle. If this command is used in any other state the command will not be accepted. In order to shut down the entire FlexRay network a higher level protocol should be used (Robert Bosch Gmbh 2006b, p104).

6.6.3 Clock Correction Failed

When the counter for the clock correction failed reaches first the ‘maximum without clock correction passive’ value the controller will transition from the NORMAL_ACTIVE to the NORMAL_PASSIVE state. If the ‘maximum without clock correction fatal’ limit is reached then the controller will transmit from either NORMAL_ACTIVE or NORMAL_PASSIVE to the HALT state.

LITERARY REVIEW

The counter allows the host to monitor the inability of a node to perform clock correction. It is incremented at the end of the odd communication cycles if either the rate or offset correction term is missing. If they are both detected then the counter is set to zero. It is also reset to zero if the controller enters the READY or NORMAL_ACTIVE state and will stop incrementing once the 'maximum without clock correction fatal' limit is reached (Robert Bosch Gmbh 2006b, p103).

6.6.4 Passive to Active Counter

This counter defines the number of cycle pairs that must have valid clock correction terms before the controller is allowed to transition from NORMAL_PASSIVE to NORMAL_ACTIVE. If this is set to zero then no transition is allowed (Robert Bosch Gmbh 2006b, p104).

6.6.5 Parity Checking

An even parity check is used to ensure integrity of data stored in the RAM blocks. The RAM blocks have a parity generator and checker attached as shown in Figure 6.4 (Robert Bosch Gmbh 2007, p7) which locally generates the parity bit and stores it with the data (Robert Bosch Gmbh 2007, p6). This is checked each time data is read from the RAM blocks but the parity checker is not able to detect which bit is incorrect and cannot repair errors. If an error is detected an associated flag is set (Robert Bosch Gmbh 2007, pp9-10).

LITERARY REVIEW

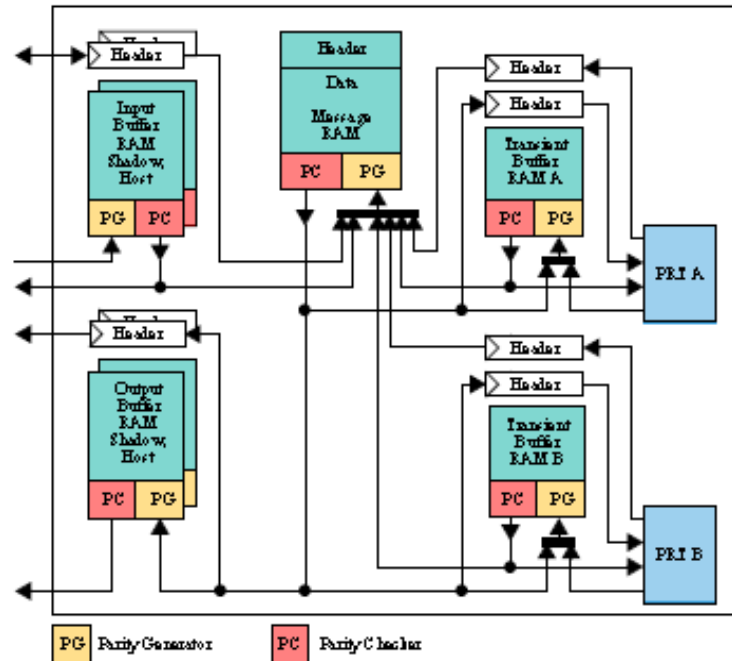


Figure 6.4: RAM blocks with local parity generators and checkers

It should be noted that the generators and checkers are not part of the RAM but lie between the core and RAM (Robert Bosch GmbH 2007, p6).

Errors can be caused by a faulty RAM cell. This may just be a temporary fault and not a permanently damaged logic cell. However as a cell will be updated at a regular interval then the problem may be self-curing or may lead to the application using an error correction routine on the detected block. If a parity error is detected the transmission of the frame will be blocked (Robert Bosch GmbH 2007, p12).

For a FIFO buffer when a parity error is detected then all data to be stored in the message buffer is lost. If the error is detected in the header section however, the FIFO needs to be reconfigured (Robert Bosch GmbH 2007, p14).

To correct the problem, if one exists, then through a word-wise read the word which is affected can be found. The error can then be bypassed through reconfiguration of the data pointer to the message buffer in question or through reconfiguration of the message buffer if the error is detected in the header section of the message RAM (Robert Bosch GmbH 2007, pp14-5).

6.7 Message Handling

A message handler controls data transfers between the input/output buffers and the message RAM as well as the message RAM and the transient RAMs. All the access to the RAMs is done as 33 bit accesses. This is because there are 32-bits of data and an additional parity bit. The use of the message handler is to avoid any possible conflict between the host and the channel protocol controllers attempting to access the message RAM (Robert Bosch Gmbh 2006b, p130).

The message RAM is scanned according to the following table, Table 6.3 (Robert Bosch Gmbh 2006b, p130).

Start of Scan in Slot	Scan for Slots
1	2...15, 1 (next cycle)
8	16...23, 1 (next cycle)
16	24...31, 1 (next cycle)
24	32...39, 1 (next cycle)
...	...

Table 6.3: Message RAM scan

The scan is terminated at the start of the network idle time section of the communications cycle even if the scan is not completed. The scan starts during slot 1 of the actual cycle starting with message RAM slot 2 with the scan of the first message slot done in the previous cycle. This check on the first slot is done in parallel with the scan of the message RAM to determine if there is a message buffer configured for slot 1 of the next cycle (Robert Bosch Gmbh 2006b, p130).

If the application needs to operate with more than 128 different messages then the static and dynamic buffers can be reconfigured during the operation of the node. To do this the header section of the respective message buffers is updated using the write header section 1 register.

If the reconfiguration of the message buffer is done before the transmission or if the buffer is updated the message is lost (Robert Bosch Gmbh 2006b, p130).

6.7.1 Host Access to the Message RAM

The host access to the message RAM occurs through the input and output buffers. It is done by the host writing the number of the target/source message buffer to the input buffer command request or output buffer command request registers (Robert Bosch GmbH 2006b, p132).

The buffers are built as a double buffer structure so that the host can access one half of the buffer while the other half (the shadow) is accessed by the message handler to allow transfer between the message RAM and input/output buffers (Robert Bosch GmbH 2006b, p132). Figure 6.5 (Robert Bosch GmbH 2006b, p132), shows the connections between the message RAM, the message handler and the Host.

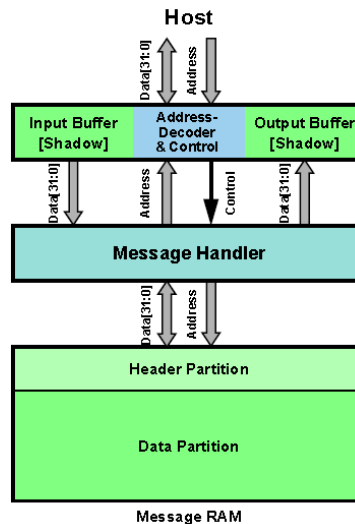


Figure 6.5: Host – message RAM interface

6.7.2 Input Buffer to Message RAM transfer

When the host writes to the target message buffer in the message RAM by using the command register, the input buffer host and shadow are swapped as in Figure 6.6 (Robert Bosch GmbH 2006b, p133).

LITERARY REVIEW

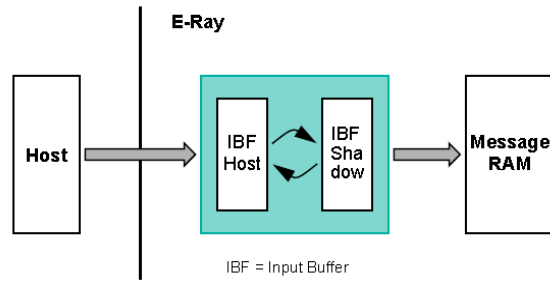


Figure 6.6: Double buffer structure input

This is in addition to the input buffer command mask and input buffer command register bits being swapped as in Figure 6.7 (Robert Bosch GmbH 2006b, p133).

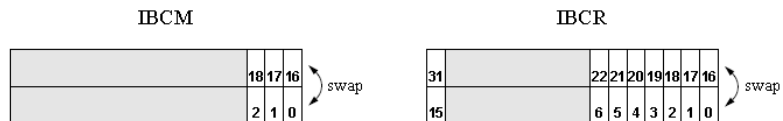


Figure 6.7: Swapping input buffer command mask & input buffer command register bits

The message RAM can now be filled with the data from the input buffer shadow while the host is still free to write to the input buffer host. When the write operations have finished and been indicated the process can start again (Robert Bosch GmbH 2006b, p133).

6.7.3 Output Buffer to Message RAM Transfer

The message RAM is read by the host writing to the output buffer command register to start transfer as defined by the output buffer command mask register. The output buffer is a double buffer, as can be seen in Figure 6.8 (Robert Bosch GmbH 2006b, p135). This is structured like the input buffer to facilitate faster data transfer.

LITERARY REVIEW

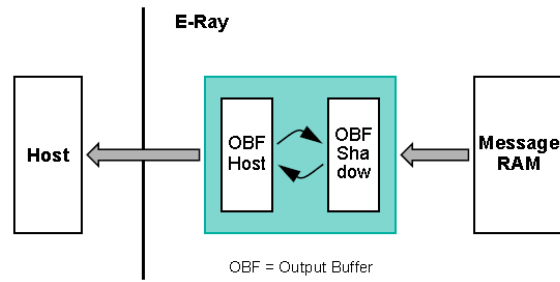


Figure 6.8: Double buffer structure output

The host and shadow buffers are swapped when a transfer request is made by the host. Some output buffer command mask and output buffer command register bits are also swapped as in Figure 6.9 (Robert Bosch Gmbh 2006b, p135).

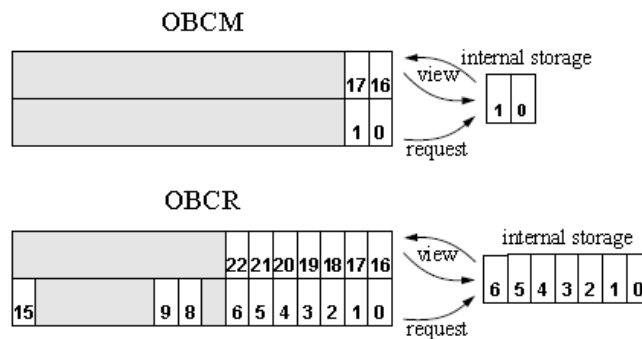


Figure 6.9: Swapping output buffer command mask & output buffer command register bits

The bits from the output buffer command mask and output buffer command register use internal storage to swap the bits as can be seen in Figure 6.9.

6.7.2 Protocol Controller Access to Message RAM

The transient buffer RAMs are used to buffer data before transfer between the two FlexRay protocol controllers and Message RAM takes place. They consist of a double buffer each so that one can be assigned to the protocol controller and the other one is accessible to the message handler. This allows for greater throughput as a message being received can be stored while the message handler writes a message to 'Transient Buffer Tx' buffer at the same time (Robert Bosch Gmbh 2006b, p138). Figure 6.10 (Robert Bosch Gmbh 2006b, p138), shows the layout of the transient buffer RAMs along with connections in and out of it.

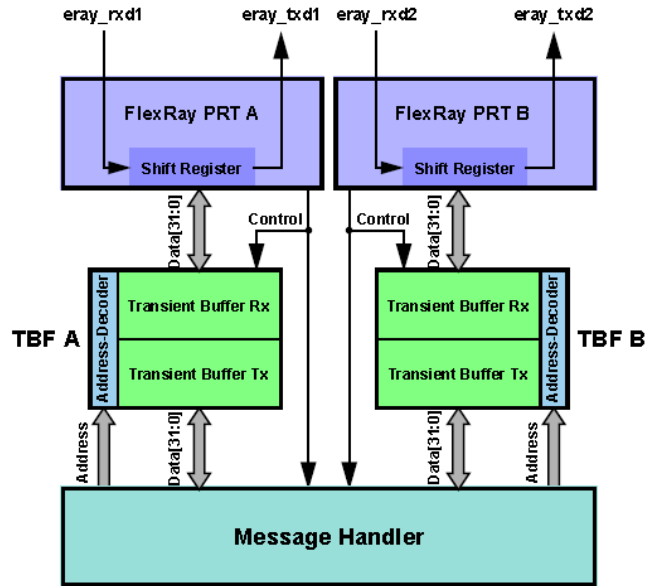


Figure 6.10: Transient buffer RAMs

6.8 Message RAM

As has been stated earlier access to the message RAM is handled by a message handler to avoid conflicts. It can store up to 128 messages depending on configuration and payload lengths.

The message RAM is organised as 2048 x 33 (= 67,548) bits as each data word is 32 bits wide with an added parity bit for protection. To achieve the flexibility that FlexRay demands the RAM is broken up into a header partition and data partition (Robert Bosch GmbH 2006b, p139). This is shown in Figure 6.11 (Robert Bosch GmbH 2006b, p139).

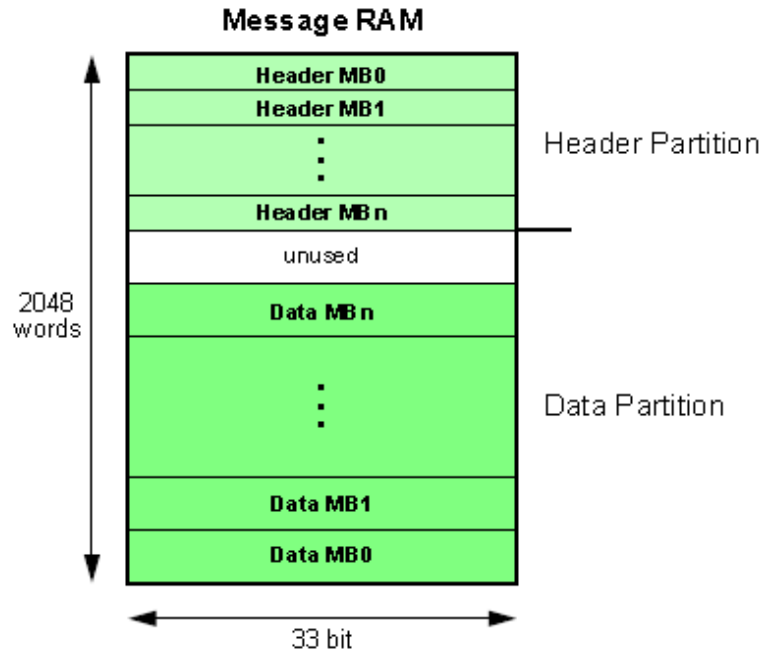


Figure 6.11: Message RAM configuration example

The header partition is used to store the header section of each message, is made up of 4 x 32+1 bit words and contains amongst other things a pointer to the data section (Robert Bosch GmbH 2006b, p139).

The data section has a maximum of 30 message buffers of 254 data bytes in length each. This can be changed depending on configurations i.e. if the data section was 128 bytes then the maximum number of message buffers would be 56 or 128 with a data section of 48 bytes (Robert Bosch GmbH 2006b, p139).

The header section is broken down as shown in Figure 6.12 (Robert Bosch GmbH 2006b, p140).

LITERARY REVIEW

Bit Word	32	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	P			M B I	T X M	P P I T	C F G	C H B	C H A	Cycle Code										Frame ID													
1	P	Payload Length Received								Payload Length Configured								Tx Buffer: Header CRC Configured Rx Buffer: Header CRC Received															
2	P			R E S	P I S	N I S	F I S	S I S	S I S	R C I	Receive Cycle Count										Data Pointer												
3	P			R E S	P I S	N I S	F I S	S I S	S I S	R C I	Cycle Count Status								F T B	F T A	M L S T	E S B	E S A	T C I B	T C I A	S V O B	S V O A	C E O B	C E O A	S E O B	S E O A	V E O B	V E O A
...	P	...																															
...	P	...																															

	Frame Configuration
	Filter Configuration
	Message Buffer Control
	Message RAM Configuration
	Updated from received Data Frame
	Message Buffer Status MBS
	Parity Bit
	unused

Figure 6.12: Header segment in message RAM

Each header word is broken down as follows (Robert Bosch GmbH 2006b, pp141-2):

Header 1:

- Frame ID
- Cycle Code
- Channel filter configuration (CHA, CHB)
- Transmit/receive configuration (CFG)
- A payload preamble transmit bit (PPIT)
- A transmit mode configuration (TXM)
- Message buffer interrupt enable (MBI)

Header 2:

- Header CRC
- Payload length configured in terms of 2-byte words
- Payload length received in terms of 2-byte words

Header 3:

- The data pointer
- Receive cycle count
- A received on channel indicator (RCI)
- A startup frame indicator (SFI)
- A sync frame indicator (SYN)
- A null frame indicator (NFI)
- A payload preamble indicator (PPI)
- Reserved bit (RES)

Message Buffer Status

This is the final header section and is made up of:

- A valid frame received on channel A (VFRA) bit
- A valid frame received on channel B (VFRB) bit
- A syntax error observed on channel A (SEOA) bit
- A syntax error observed on channel B (SEOB) bit
- A content error observed on channel A (CEOA) bit
- A content error observed on channel B (CEOB) bit
- A slot boundary violation observed on channel A (SVOA) bit
- A slot boundary violation observed on channel B (SVOB) bit
- A transmission conflict indication on channel A (TCIA) bit
- A transmission conflict indication on channel B (TCIB) bit
- An empty slot on channel A (ESA) bit
- An empty slot on channel B (ESB) bit
- A message lost (MLST) bit
- Cycle count status
- A received on channel indicator status (RCIS)
- A startup frame indicator status (SYNS)
- A null frame indicator status (NFIS)
- A payload preamble indicator status (PPIS)
- A reserved bit status (RESS)

LITERARY REVIEW

The data partition starts after the last word of the header section and the data sections are stored as in Figure 6.13 (Robert Bosch Gmbh 2006b, p143). This example shows a data section with an odd number of 2-byte words and thus the last 16-bits in the 32-bit word are unused (Robert Bosch Gmbh 2006b, p143).

Bit Word	32	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
...	P	unused								unused								unused								unused							
...	P	unused								unused								unused								unused							
...	P	MBn Data3								MBn Data2								MBn Data1								MBn Data0							
...	P							
...	P							
...	P	MBn Data(m)								MBn Data(m-1)								MBn Data(m-2)								MBn Data(m-3)							
...	P							
...	P							
...	P							
...	P	MB1 Data3								MB1 Data2								MB1 Data1								MB1 Data0							
...	P							
...	P	MB1 Data(k)								MB1 Data(k-1)								MB1 Data(k-2)								MB1 Data(k-3)							
2046	P	MBO Data3								MBO Data2								MBO Data1								MBO Data0							
2047	P	unused								unused								MBO Data5								MBO Data4							

Figure 6.13: Data partition in message RAM example

The beginning and end of the data section are determined by the data pointer and payload configuration length configured in the header section. This makes it a flexible system and suitable for FlexRay (Robert Bosch Gmbh 2006b, p143).

6.8.1 Message RAM Configuration

To define how many buffers are assigned to the static and dynamic segments as well as the FIFO, the message RAM configuration register should be configured by the application programmer (Robert Bosch Gmbh 2006e, p6).

Some of the bits to be configured in this register are the first dynamic buffer, first buffer of the FIFO and the last configured buffer (Robert Bosch Gmbh 2006e, p6). These are also known as FDB, FFB and LCB respectively and referred to as such in Table 6.1 at the start of this chapter.

To define the size of the FIFO the FFB bits and LCB bits are used and if a dynamic buffer or buffers are configured then the value stored in the FFB bits should be greater than that of the value in the FDB. To disable the dynamic buffers the FDB value should be greater than 128 and to disable the FIFO the FFB value should also be greater than 128. LCB must also be greater than FDB and FFB as it is not possible to put the

LITERARY REVIEW

FIFO header section or dynamic header section before the static section. It should be noted that there is no checking in place that the configuration is valid and as such it is up to the programmer to ensure the setup meets the specifications (Robert Bosch GmbH 2006e, p8).

The first buffer, buffer 0, can be used to store a startup frame, sync frame, a single slot frame or a normal frame. This is defined by various registers. If the value of the startup, sync or single slot frame is different for channels A and B then the second buffer can be assigned to channel B and the first buffer is used for channel A (Robert Bosch GmbH 2006e, pp8-9).

6.9 Filtering and Masking

Filtering is done by comparing the configuration of message buffers against slot, cycle and channel ID values. A message buffer will only be updated or transmitted if matching occurs (Robert Bosch GmbH 2006b, p121). The combinations for filtering that are permitted are (Robert Bosch GmbH 2006b, p121):

- Slot counter & channel ID
- Slot counter , cycle counter & channel ID

6.9.1 Slot Counter Filtering

The header section of the transmit buffer and the receive buffer holds a frame ID. The frame ID from the message buffer is compared to the slot counter values in order to assign transmit and receive buffers to the slot. If two message buffers have the same frame ID then the lowest message buffer will be used if the cycle counter filter is the same (Robert Bosch GmbH 2006b, p121).

6.9.2 Cycle Counter Filtering

Again each message buffer will hold a cycle set field in the header section. If a match is observed this is due to one of the elements of the cycle set being matched. The set of cycle numbers belonging to the cycle set is determined as shown in Table 6.4

LITERARY REVIEW

(Robert Bosch Gmbh 2006b, p122) with an example shown in Table 6.5 (Robert Bosch Gmbh 2006b, p122).

Cycle Code	Matching Cycle Counter Values
0b000000x	all Cycles
0b000001c	every second Cycle at (Cycle Count)mod2 = c
0b00001cc	every fourth Cycle at (Cycle Count)mod4 = cc
0b0001ccc	every eighth Cycle at (Cycle Count)mod8 = ccc
0b001cccc	every sixteenth Cycle at (Cycle Count)mod16 = cccc
0b01ccccc	every thirty-second Cycle at (Cycle Count)mod32 = cccccc
0b1cccccc	every sixty-fourth Cycle at (Cycle Count)mod64 = ccccccc

Table 6.4: Cycle set definition

Cycle Code	Matching Cycle Counter Values
0b0000011	1-3-5-7- ... -63 ↓
0b0000100	0-4-8-12- ... -60 ↓
0b0001110	6-14-22-30- ... -62 ↓
0b0011000	8-24-40-56 ↓
0b0100011	3-35 ↓
0b1001001	9 ↓

Table 6.5: Examples of cycle sets

Received messages are only stored if the cycle counter value during which the message is received matches an element of the cycle set and the other filtering criteria are met (Robert Bosch Gmbh 2006b, p122).

Transmit frames are transmitted on the desired channel or channels when an element in the cycle set matches the cycle counter value and other filtering criteria are met. It should be noted that sharing of a static time slot, across a number of different nodes, by using cycle counter filtering is not allowed (Robert Bosch Gmbh 2006b, p122).

6.9.3 Channel ID Filtering

Each message buffer has a filtering field for the channel in the header section that uses 2-bits. For receive buffers it acts as a filter while transmit buffers use it as a

LITERARY REVIEW

control field (Robert Bosch Gmbh 2006b, p123). This can be seen in Table 6.6 (Robert Bosch Gmbh 2006b, p123).

CHA	CHB	Transmit Buffer transmit frame	Receive Buffer store valid receive frame
1	1	on both channels (static segment only)	received on channel A or B (store first semantically valid frame, static segment only)
1	0	on channel A	received on channel A
0	1	on channel B	received on channel B
0	0	no transmission	ignore frame

Table 6.6: Channel filtering bit configurations

The frames to be transmitted will be sent out according to this configuration if other filtering criteria are met. Received frames will be stored if they are received on the correct channel as specified by the table and other filtering criteria are met. Only frames transmitted or received during the static segment of the communication cycle are allowed to be configured for both channels. When dynamic segment frames that are set up so both bits of the channel ID filter are set as a logic one then this will be treated as though the bits were both set as a logic zero, i.e. 'no transmission' (Robert Bosch Gmbh 2006b, p123).

6.9.4 FIFO Filtering

The filtering for the FIFO is different to the message RAM filtering. This uses a rejection filter and filter mask. The filter consists of a channel, frame and cycle counter filter. The cycle counter filter determines the cycle set to which the other filtering is applied and all other frames in other cycles are rejected. A valid received frame is stored in the FIFO if the rejection filter and rejection filter mask do not correspond to the frame and there is no matching receive buffer (Robert Bosch Gmbh 2006b, p123).

6.10 FIFO

The FIFO is a First-In-First-Out cyclical buffer. The buffers belonging to it are found one after another in the register map. The message RAM configuration register defines the first and last register of the FIFO using the values in the FFB bits and LCB

LITERARY REVIEW

bits with a maximum of 128 buffers for the FIFO (Robert Bosch Gmbh 2006b, p128; Robert Bosch Gmbh 2006f, p6).

The FIFO is used to store incoming frames that do not have a dedicated receive buffer. It also treats null frames that are not filtered out as data frames and stores them (Robert Bosch Gmbh 2006b, p128; Robert Bosch Gmbh 2006f, p6).

There are two index registers associated with the FIFO, and these are the put next index (PIDX) register and the get next index (GIDX) register. When a new message is to be stored in the FIFO it is stored in the buffer pointed to by the PIDX register and this is then incremented. The GIDX register is used to point to the next buffer which is to be read and it increments when a buffer is read (Robert Bosch Gmbh 2006b, p128; Robert Bosch Gmbh 2006f, p7).

If the PIDX register reaches the value of the GIDX then the FIFO is filled. If a new message is written before the oldest message is read this will cause an overrun flag to be set (Robert Bosch Gmbh 2006b, p128; Robert Bosch Gmbh 2006f, p7). Three of the possible states the FIFO can be in are shown in Figure 6.14 (Robert Bosch Gmbh 2006b, p129) below.

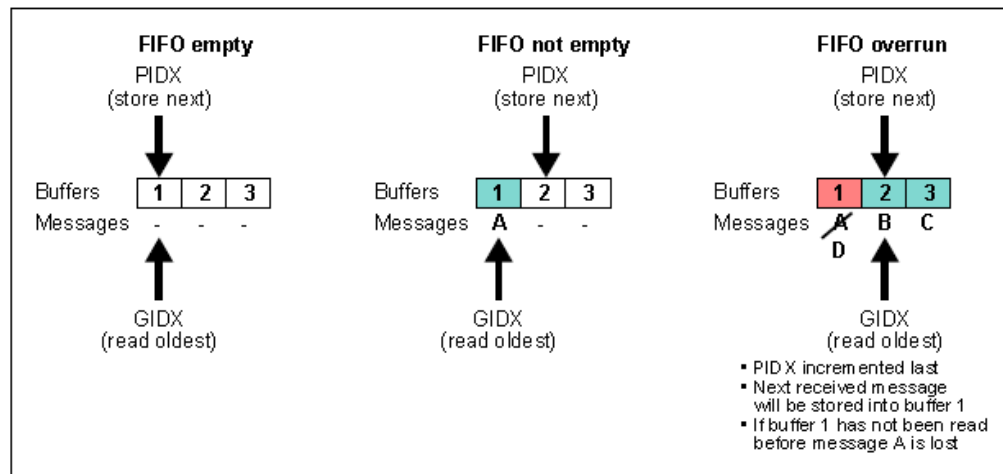


Figure 6.14: Empty, not empty and overrun states

When the PIDX and GIDX registers differ a FIFO not empty status is detected (Robert Bosch Gmbh 2006f, p7).

To access the FIFO outside the CONFIG and DEFAULT_CONFIG states involves the host triggering a transfer from the message RAM to the output buffer by writing the first message buffer of the FIFO to the output buffer command request

register. The message handler then will transfer the message pointed to by the GIDX register to the output buffer and the GIDX register is incremented (Robert Bosch GmbH 2006b, p129).

6.11 Packaging

As the E-Ray IP-module is described in VHDL there is no specific packaging designed. It is left to the company who buys the license for the E-Ray module to package it and sell it on. This means that some features found in a certain E-Ray chip may not be available in another by a different manufacturer or indeed the same manufacturer.

For instance the Fujitsu MB88121 is a 64 pin low profile quad flat pack chip and amongst other things also contains a DMA support unit and a Serial Peripheral Interface (SPI) interface possibility (Fujitsu Microelectronics Europe GmbH 2007). Both of these additions were implemented by the manufacturer as the E-Ray module does not have specifications for either.

6.11.1 VHDL

VHDL is a hardware description language. It stands for Very High Speed Integrated Circuit (VHSIC) Hardware Description Language (HDL). It was developed in the 1970s and 1980s as part of the U.S. Department of Defence VHSIC program. It was initially intended to be used to describe complex circuits. This was to help make the designs of hardware modules easier to understand by different contractors. It was also designed to allow simulation of the circuit designs. In 1996 IEEE 1076.3 became a VHDL synthesis standard based on the IEEE 1164 and IEEE 1076 standards. Verilog is another type of HDL used whose programming syntax is considered less verbose but lacks features when compared to VHDL (Shakill 1996).

6.12 Conclusion

The E-Ray IP-module developed by Bosch is a very useful and flexible device. It is fully compliant with the FlexRay protocol v2.1 and allows customer specific CPUs to connect to it. This makes it one of the most common FlexRay protocol IP modules available.

It can be seen that there are a number of aspects that must be defined in order for the E-Ray chip to function as intended. These setup parameters could have an enormous effect on the performance of the node. A number of errors could be present in a system if the node is not setup correctly. These include missed deadlines or loss of data due to data being stored incorrectly or not accessed in time. It is therefore useful to be able to test the configuration. The research outlined in this thesis makes this process easier.

6.13. References

Berwanger, J., Schedl, A. and Peller, M (2004) BMW – First Series Cars with FlexRay in 2006, Automotive electronics + systems Special Edition, Development Solutions 19 for FlexRay ECUs, 6-8

BMW Manufacturing Co. (2006) THE NEW BMW X5

Perfect Blend of Driving Dynamics, Functionality and Exclusivity [press release], 8 August, available:

http://www.bmwusfactory.com/media_center/releases/release.asp?intReleaseNum=209&strYear=2006 [accessed 2 October 2007].

FlexRay Consortium (2005) FlexRay Communication System Protocol Specification, Version 2.1 Revision A, Stuttgart: FlexRay Consortium GbR.

FlexRay Consortium (2006) news – FlexRay Newsletter 03/2006 [online], available: http://www.flexray.com/news/FlexRay_Newsletter2006_03.pdf [accessed 22 October 2007].

LITERARY REVIEW

Fujitsu Microelectronics Europe GmbH (2005) Automotive Solutions CMOS FlexRay ASSP MB88121/MB88121A/MB88121B/MB88121C preliminary datasheet, Revision 1.0, Langen: Fujitsu Microelectronics Europe GmbH

Robert Bosch GmbH (2006a) Product Information E-Ray – The FlexRay Communication Controller – IP-Module, Reutlingen: Robert Bosch GmbH.

Robert Bosch GmbH (2006b) E-Ray FlexRay IP-Module User's Manual, Revision 1.2.3, Reutlingen: Robert Bosch GmbH.

Robert Bosch GmbH (2006c) E-Ray FlexRay IP-Module Application Notes AN001 Wakeup, for IP Revision 1.0, Reutlingen: Robert Bosch GmbH.

Robert Bosch GmbH (2006d) E-Ray FlexRay IP-Module Application Notes AN002 Startup, for IP Revision 1.0, Reutlingen: Robert Bosch GmbH.

Robert Bosch GmbH (2006e) E-Ray FlexRay IP-Module Application Notes AN003 Message RAM Configuration, for IP Revision 1.0, Reutlingen: Robert Bosch GmbH.

Robert Bosch GmbH (2006f) E-Ray FlexRay IP-Module Application Notes AN004 FIFO Function, for IP Revision 1.0, Reutlingen: Robert Bosch GmbH.

Robert Bosch GmbH (2007) E-Ray FlexRay IP-Module Application Notes AN005 Handling of Parity Errors, for IP Revision 1.0.1, Reutlingen: Robert Bosch GmbH.

Skahill, K. (1996) VHDL for Programmable Logic, California: Addison-Wesley Publishing Company, Inc.

Chapter 7 . Discrete Event Simulation

7.1 Introduction

Testing of systems can be costly in both time and money. If a new product is being developed and a prototype is run for the first time it may not work as expected. This can lead to the design team spending time just diagnosing problems, especially if the system is large and complicated. Simulating a system before a real world system is developed has become popular due to the nature of simulation; it can be repeated many times and the data obtained from it easily collected. If a problem arises a computer model can be changed quickly and cheaply when compared to a prototype. Banks et al. (2001, p3) sum up simulation as the imitation of a real-world process or system over time. This can be done by hand or on a computer, with an artificial history of the system being logged. From this history observations on how the system works and how it behaves can be observed.

Woolfson and Pert (1999, pv) wrote that experiments controlled by computer, with the data logged and analysed by a computer are allowing an increase in the range and the accuracy of what can be done.

However simulation is not only restricted to new products but can also be applied to existing products; this is because technically simulation is used to produce results from a model without experimenting with a real-world system (Ripley 1987, p1). It can be used by an engineer or scientist to get a better understanding of the behaviour of a system and to pinpoint any areas for improvement. It should also be stated that simulation is not necessarily restricted to engineering and scientific applications. Simulation has contributed to problem solving in the areas of economics, management, as well as in social and behavioural sciences (Neelamkavil 1987, pxv.). This chapter is written in terms of discrete event simulation (DES) specifically.

7.2 Systems

It is important when introducing the topic of system simulation to first understand what is meant by the term 'system'. A system is the key concept of simulation in terms of understanding what is to be achieved. It is defined as a collection of entities (elements of a real world system, such as parts of a cars engine) that interact with each other in a manner to accomplish some goal (Law and Kelton 1982, p2).

The definition above is a very good general definition of what a system is. In a simulation setting a system is defined by the particular area of study. A system describing a communication protocol in an automotive application will be defined by what particular protocol is used and what applications it is designed to run. It is not necessary to incorporate the type of car or who will be driving the car in the system. It is important however to include any application using the system and the communication method as they work together to form the communication system.

In the study of a system a few terms must be defined. These will help us clarify the idea of a system. The terms are as follows (Banks et al. 2001, p10):

- Entities : These are objects of interest in the system i.e. customers in a bank.
- Attributes: These are defined as properties of entities i.e. bank balance.
- Activities: These are time periods of specific length i.e. checking a bank balance.

It is the case that the collection of entities that might encompass a system used in one study might just be a subset of the system used in another study (Law and Kelton 1982, p2).

Entities will also have a particular set of 'states' associated with them. For instance in a communications application the number of frames a node must send, the time it takes to send a single frame or the number of frames that can be sent by a particular node at any given time. Therefore states can be seen as variables describing the system at any stage of the study (Banks et al. 2001, p10).

Events can occur within a system or come from the environment outside the system and change the state of the system. In the communications applications example an event could be the completion of transmission of a frame from a node or an error check which returns no errors, and so the number of frames to be transmitted will decrease. The generation of new data to be transmitted from a sensor could also be an event.

7.2.1 Continuous and Discrete Systems

Systems are also broken down into discrete and continuous system types. Figure 7.1 shows a discrete system variable and Figure 7.2 shows a continuous system variable (Banks et al. 2001, p12).

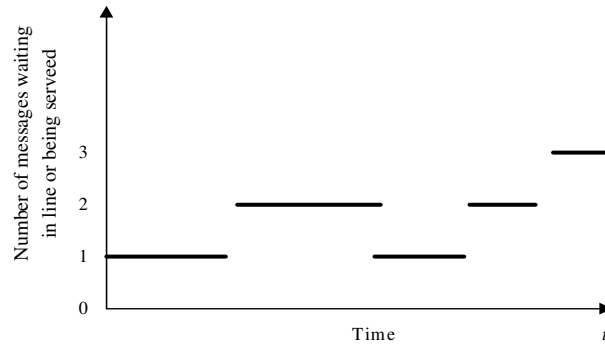


Figure 7.1: Discrete-system state variable

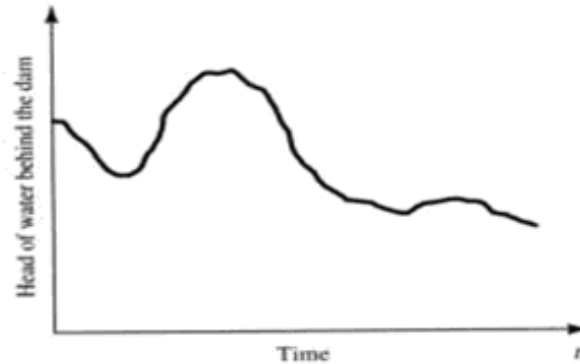


Figure 7.2: Continuous-system state variable

Discrete system simulation involves modelling a system as it changes over a period of time. The state variables will change at defined points in time (Law and Kelton 1982). To describe this Law and Kelton (1982, p4) described a barber shop or information desk at an airport. In this example customers arriving have to wait their turn in a queue and this happens at discrete points. After each customer is serviced the next customer can be serviced and so the number of customers in the queue will decrease but again at discrete points in time.

A continuous system is one where the variables change in an analogue fashion. This means that the variables change continually with respect to time. An example of this (Banks et al. 2001, p10) would be the level of water behind a dam. The water level

behind the dam will be continually changing as it is subject to weather i.e. precipitation and evaporation, which will cause the level of the water to rise and fall. The water level will also be subject to the operation of the dam and how much water is allowed to pass in the production of electricity.

In practical applications it is not always convenient to model a system as either discrete or continuous since real world systems rarely have attributes that are wholly discrete or continuous. Usually a system is a combination of both but one definition will encapsulate the majority of the operation of the system enough to classify it as either discrete or continuous (Banks et al. 2001, p10). However it is sometimes necessary to construct a system with aspects of both, and these simulations are called 'combined discrete-continuous' simulations (Law and Kelton 1982, p47).

7.3 Simulation Process

The steps in a simulation study are shown in Figure 7.3 (Banks et al.2001, p16).

The main steps can be summarised as follows:

1. Problem identification/formulation
2. Building of a model
3. Model verification
4. Model validation
5. Experimental tests
6. Results Analysis

These steps are discussed in Chapter 10. The problem identification/formulation stage will identify the need for a simulation model to be constructed and define the problem statements. This is an important stage of simulation but outside the scope of this chapter as it is the job of the person conducting a study to identify the need to simulate a system.

LITERARY REVIEW

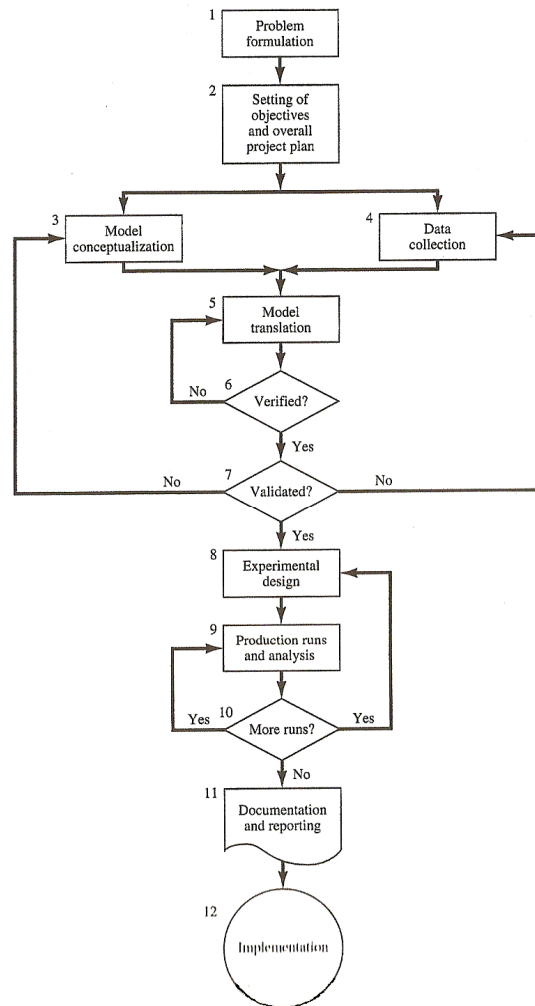


Figure 7.3: Simulation study steps

7.4 Building Models

To simulate a system requires the building of a model of the system. This is due to the fact that it is not always possible to easily change a system and monitor the changes. The system under investigation could be an existing real-world application that will be modified or a new design for a product in development. In both cases it could be costly to build a prototype of the new system just for a major design flaw to be present.

The model of a system should be comprehensive enough to allow conclusions to be drawn from the simulation output data. The idea of a model is that it is a simplified version of the system under investigation (Banks et al.2001, p13). As such a model is described as having entities, attributes and activities just as a system. The

LITERARY REVIEW

difference between a real-world system and a model is that a model only contains components that are of interest to the study (Banks et al. 2001, p13).

Models can be as detailed as the designer wants them to be. Woolfson and Pert (1999, p1) give an example of a child's toy car. In this example they describe how a simple model may be made out of clay, with disks for wheels and lines etched in the side to represent doors of the car. This will do as a child's toy car but it does not have all the features a real car does. No matter how many extra features are added to make it more like a car, the only truly accurate representation of a car is a real car. Therefore the designer must include only those attributes that are relevant to the area of study.

Models fall under different classifications. Models may be mathematical or physical, with mathematical models being broken down into further subcategories. A simulation model is a type of mathematical model that can be Monte Carlo simulation (static), dynamic, deterministic or stochastic (random), discrete or continuous (Banks et al. 2001, p13). Static models deal with simulations at a specific point in time while a dynamic model is used for simulating systems as they change over time. A system that has behaviour in terms of fixed inputs and that will produce a single set of outputs is described as deterministic. A random model is used where the inputs are random and as such will produce a random set of outputs. Discrete models have inputs that change at given points in time while a continuous model has an ever changing input (Banks et al. 2001, p12). Figure 7.4 shows one possible breakdown of the different types of simulation models.

LITERARY REVIEW

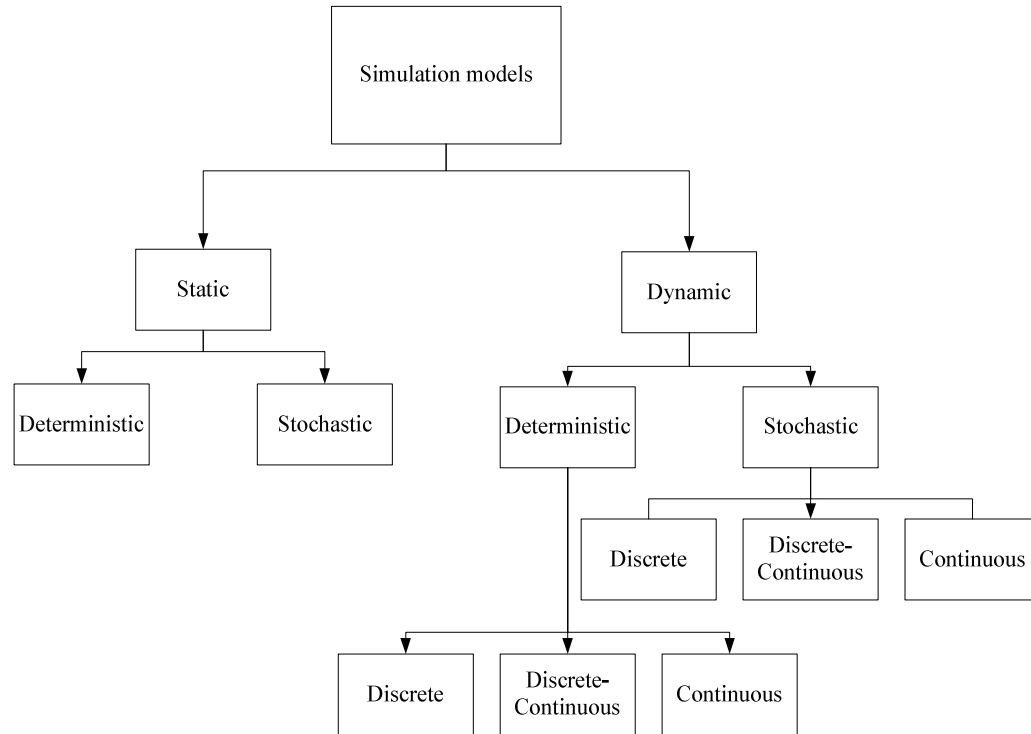


Figure 7.4: Simulation models

7.4.1 Hardware in the Loop Simulation

A consideration when building a simulation model is that of using 'Hardware in The Loop Simulation' (HILS). In many systems an output is not directly related by a simple mathematical formula. For instance a simple system circuit is that of a potentiometer connected to an Analogue to Digital Converter (ADC) and an LCD digital display. As the position of the potentiometer changes so too does the ADC output value which is then displayed on the LCD screen. This is easily tested by simply turning the potentiometer and checking the display. Real world systems may contain this simple circuit as part of an overall more complicated system.

The success of complicated systems can be dependant on adequate testing. Testing becomes more and more complicated for real-word systems where multiple embedded systems are involved. The output of a testing procedure may be formed by a microcontroller that receives information form a number of different sources. To adequately test such a system would mean stopping the test at a number of points in time and observing if the correct output was produced. This can become difficult and time consuming as each output must be verified based on every possible input that is

LITERARY REVIEW

received from the different system elements. It is made even more difficult if each subsystem is not easily stopped to allow the inputs and outputs to be checked. This will mean testing must be carried out in real time (Gomez 2001).

HILS is a technique that allows a system to be ‘fooled’ into thinking it is operating in the real world. An example of such a system is a vehicle, missile or aeroplane as all of these examples will receive multiple inputs and an onboard microcomputer should produce a desirable output. A HILS system will therefore fool an aeroplane into thinking its flying or a vehicle into thinking that the engine is running and that the vehicle is in motion. A HILS simulator will then need to provide all inputs for the system to allow all the subsystems of the overall system under test to function correctly (Gomez 2001). This can allow a developer to by-pass the model building stage if done correctly.

Gomez (2001) highlights the difference between a HILS approach and that of simulating a control algorithm using MATLAB. A test of the control algorithm using MATLAB will be run using a PC with ‘faked’ inputs to the system and observe the output using the same PC. The output obtained will be represented as a graph or set of numbers but it cannot produce a real-world hardware signal. The control algorithm will also be run in an environment that does not represent real-world time. Instead the control algorithm will be simulated based on a simulation clock that may have correlation to real-world time. A HILS test has the advantage of allowing the real hardware to be tested while running the real-world software and in real time. Figure 7.5 (Xun et. al. 2008) shows an example of a flight control HILS system. In this system the simulation computer controlling the mathematical models provides the external environment for a flight control system.

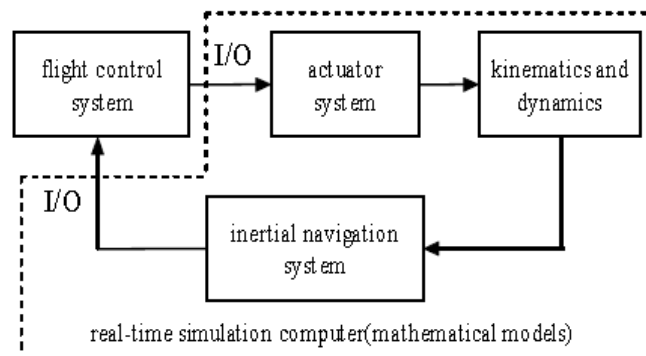


Figure 7.5: Flight control HILS system

LITERARY REVIEW

The advantages of using HILS relates to a number of factors such as high precision. For systems where safety could be an issue (such as passenger safety in a vehicle or aircraft), testing the different systems requires a high precision and confidence in the different component outputs. HILS provides a cheaper alternative to testing a complete aircraft or vehicle system (Xun et. al. 2008). HILS will then also allow any prototype of a system to be built at a later stage in the development cycle (Zhu et. al. 2009). Hwang et. al. (2006) highlight how a vehicle or aircraft may be tested in a safer manner by using HILS. When a vehicle is being tested using HILS, it may be tested for an output during dangerous manoeuvres that could be dangerous for a test driver or pilot to perform before it is tested in the real world. This could lead to a decrease in accidents where a person's life may be put at risk. Safety is listed as an advantage by Applied Dynamics International (2007) along with cost and time benefits. Using HILS they emphasise how a reduction in cost and time can be achieved. This is as parallel tests of different subsystems can be carried out in a timely fashion.

HILS does not suit every test however and the type of testing will indicate if HILS is suitable. One major factor when considering if HILS is suitable is that little information may be extracted from a device if the system does not behave as expected. HILS does not let you analyse the internal behaviour of a system at run time only the outputs of the system under test (Gomez 2001).

7.5 Validation & Verification

It is important to first distinguish between the verification and validation of a model. Verification of a model is determining if the model of the system works as intended. The validation of the model is a process to determine if the simulation model is an accurate representation of the system under investigation.

Verification of the model will usually consist of debugging a computer program if the simulation is computer based. It focuses on "building the model right" (Banks et al. 2001, p367). Validation is more concerned with "building the right model" (Banks et al. 2001, p367).

Once the model of a system has been constructed it cannot be used immediately for measuring data. It is important to first calibrate the model. A simulation model

LITERARY REVIEW

might be acting in a very similar manner to the real system; however its exact behaviour may not truly reflect the system aspects that the model is intended to simulate. The model may fail to produce accurate information if it has not been calibrated and validated (Mitrani 1982, p41).

Figure 7.6 (Banks et al. 2001, p369), shows the ongoing model-building process with the use of validation and verification.

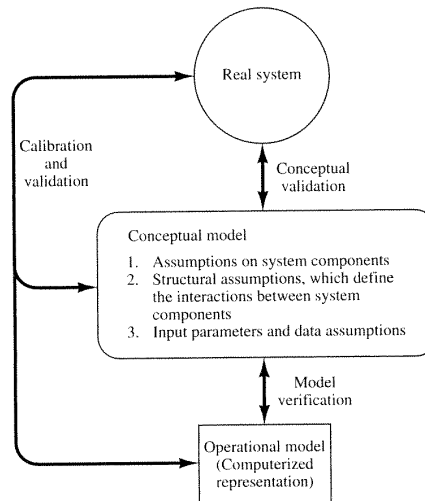


Figure 7.6: Model building, verification and validation

To carry out the validation it can be seen from Figure 7.6 that calibration is used. Essentially this means that the model is compared to a real world system and adjustments made to the model to more accurately reflect the real world. This process is then repeated until an acceptable level of accuracy is achieved (Banks et al. 2001, p369). One possible way in which a model may be validated incorrectly is if only one data set is used. In this case the model may be able to accurately represent the system for this set only. It is therefore important to validate the model using a number of different data sets (Banks et al. 2001, p375).

7.6 Tests and Analysis

Once a model has been constructed and validated then the next step of simulation is to run the experiment and collect the data for analysis. The data collected should be relevant to the experiment under study and the method of collection should

LITERARY REVIEW

lead to easy analysis. This may seem trivial with the ability of modern computers to handle vast amounts of data along with the ease of creating charts from the data. However time should be spent when designing the system to determine if the data being collected is relevant. Also time should be given over to consider if there is any data not collected that could lead to better understanding of the system and what is happening. It should be noted that extra readings might be needed when validating the model to ensure all sections perform as intended, but these readings may not be of any use to the actual experiment under consideration.

Analysis of the data is also important. Once data is collected it should be presented to anyone who wishes access to the data to easily draw their own conclusions. For this reason the use of charts can be useful. It is rarely possible to be able to draw conclusions from a long list of numbers. This means that data should not only be represented this way but also in a visual way in the form of charts and graphs.

At the analysis stage it might be discovered that not enough information was collected and more runs of the simulation with different variables may be needed to gain a clearer view of the systems behaviour and/or performance (Banks et al. 2001, p18).

7.7 Simulation of Queues, Statistics and Random Numbers

The modelling of events is an important part of simulation. This can be in the form of a queue of people waiting to be served in a bank or shop, instructions waiting to be processed on a processor or aeroplanes waiting to take off from an airport. If the server or servers, i.e. bank tellers, CPU's or runways for the same of examples given above, are currently busy then a queue is formed.

A queuing method is then generally needed to handle the arriving customers (a customer is any entity that is seen to be seeking service from a system (Banks et al. 2001, p205)). For instance in an airport aeroplanes arriving and departing will have a fixed schedule and so how the queue forms and acts may be easily understood. In a supermarket customers arriving at a checkout may seem to choose a random queue to enter based on their perceptions of what would be the quickest line to join; the arrival of customers in the shop may also be of a random nature.

LITERARY REVIEW

For these reasons simulation may require the modeller to be knowledgeable in the area of statistics and random numbers. It should also be noted that a stochastic model may even be used to analyse a deterministic system (Ripley 1987, p1). Stochastic systems can be quite difficult to analyse and even more so as the system under investigation becomes larger. For these reasons methods to evaluate these problems have been developed. Included in these are the Monte Carlo method and the Markov process.

7.7.1 The Monte Carlo Method

The Monte Carlo method was developed during the Second World War by Stanisław Ulam and John von Neuman. It was designed to quickly solve problems they were coming across while developing the atomic bomb. The name was derived due to the fact that random numbers were used to determine variables. This can be compared to many gambling scenarios such as roulette tables or throwing dice (Woolfson and Pert 1999, p22). This makes the simulation of large systems easier than would otherwise be possible. The use of Monte Carlo methods also allows different configurations of the system to be run more quickly. As such a whole system and not a subsection of the system may therefore be described where other methods may prove too costly or difficult to use (Woller 1996).

In summary Monte Carlo techniques are any techniques where random numbers and probability are used to solve a scenario. It is used in many different applications such as nuclear physics or traffic control problems. Within different disciplines there can be many different techniques and subsets of Monte Carlo simulation (Woller 1996).

7.7.2 The Markov Process

Didkovsky (1996) defines the Markov process as a way of determining the likelihood of a random dependant event occurring. The likelihood of some random events can be influenced by previous events. Didkovsky (1996) explains that a coin toss cannot be modelled as a Markov process as the coin has no memory of what occurred before. However a communicating node in an automotive network may respond to a message it has received and can be modelled as a Markov model. Weisstein (2007a) defines the Markov process in the following way:

LITERARY REVIEW

“A random process whose future probabilities are determined by its most recent values. A stochastic process $x(t)$ is called Markov if for every n and $t_1 < t_2 \dots < t_n$, we have

$$P(x(t_n) \leq x_n | x(t_{n-1}), \dots, x(t_1)) = P(x(t_n) \leq x_n | x(t_{n-1})).$$

This is equivalent to

$$P(x(t_n) \leq x_n | x(t) \text{ for all } t \leq t_{n-1}) = P(x(t_n) \leq x_n | x(t_{n-1}))”$$

7.7.2.1 Markov Chains

A Markov chain is a series of random states that is dependant on probabilities of transitioning from one state to another state (Carter 1996). Markov Chains can give a good representation of a system over a small sample, however on large systems they may make little sense (Didkovsky 1996).

To calculate a Markov chain a matrix is often used. In this form the elements of the matrix represent the possibility of transitioning from one state to another. This can be seen below where $P_{1,2}$ represents a transition from state one to state two, $P_{2,1}$ represents a transition from state two to state one etc.

$$X = \begin{pmatrix} P_{1,1} & P_{1,2} \\ P_{2,1} & P_{2,2} \end{pmatrix}$$

In “An Introduction to Computer Simulation” Woolfson and Pert (1999, p138) give three conditions to be satisfied to generate variables to settle down to the required distribution:

1. The sum of the elements in a row should equal 1.
2. $P(x_i)P_{i,j} = P(x_j)P_{j,i}$, where $P(x_i)$ is the required probability for the variable x_i .
3. The elements allow all the variables to be accessed.

If the process is useful then successive operations will generate all possible values of the variable and the probability distributions should eventually settle down to the required distributions and remain there (Woolfson and Pert 1999, p137).

7.7.3 Queuing Theory

Queuing Theory falls into two types of categories (Slater 2000):

LITERARY REVIEW

1. Open Queuing Network.
2. Closed Queuing Network.

The first type (open queuing network) is where an external system generates customers to be processed by the network and then arrives at an external source. The closed queuing network has a fixed population and this population cannot leave the system. Figure 7.7 (Slater 2000) shows an open queuing system and Figure 7.8 (Slater 2000) shows a closed queuing system. The yellow circle represents a customer source and the purple inverted trapezoid is a sink. The blue boxes connected to the red dots are service centres. Each of the black lines indicate a possible path an entity may travel down.

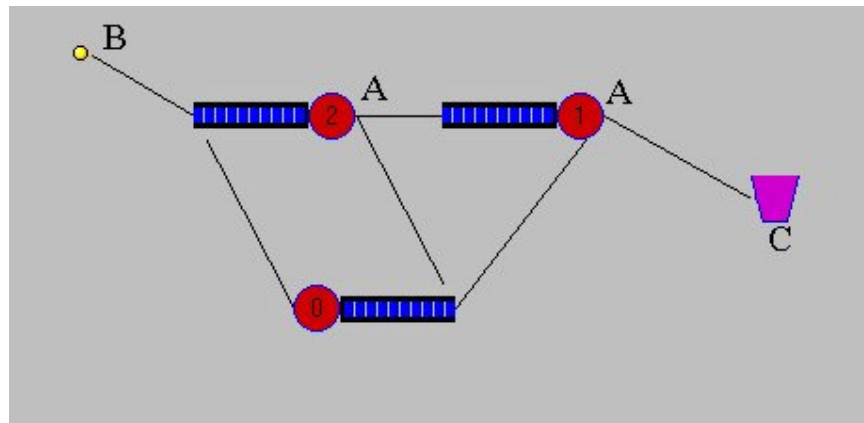


Figure 7.7: Open queuing network

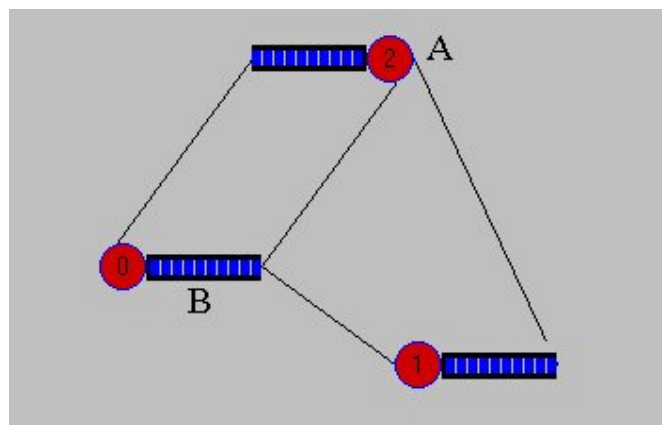


Figure 7.8: Closed queuing network

When performing queuing analysis there are a few factors to be addressed, such as how the customers are arriving and how long it takes to process a customer. From this a

LITERARY REVIEW

standard notation for queuing theory has been developed. To characterise a queuing system the Kendall notation is often used (Willing 1999, p5). The Kendall notation is of the form:

$$A/B/m/N - S$$

Where A is the distribution of the arrival of customers, B is the distribution of the service times, m is the number of servers and N is taken as ∞ if not given but represents the maximum size of the waiting line. S is optional but represents the service discipline. This is taken as First-In-First-Out (FIFO) if not given. A and B can represent a Markov (M), deterministic (D), Erlang-k (E_k), Hyper-k (H_k) or General (G) distribution. S can be a FIFO, Last-In-First-Out (LIFO), random, Round Robin or priority service discipline. An example of the above would be an M/M/1 queuing system. This is a Markov distributed system, with one server and a FIFO service discipline.

There is also notation for various aspects of the system such as the number of customers in a closed network (K), customer number (C_n) and the arrival rate to node i (A_i) amongst others as these are important aspects of the queue. Also the type of queue (FIFO, LIFO etc.) is an important aspect of system as it will define how entities are routed through a network, and as such needs to be carefully considered and represented.

A queuing theory model will allow a systems analyst to obtain a number of performance metrics. Some metrics that could be obtained are: average queue length, average queue wait time and server throughput rates. Analysis of these metrics may allow a system analyst to identify problems in a system such as bottlenecks. Steps may then be implemented to improve the performance of the system.

7.8 Simulation Software

Software used to create models and run simulations have varied from general purpose programming languages, such as C++, to general purpose simulation software (GPSS) that is solely designed to carry out simulation. Some of these GPSS systems incorporate a graphical user interface (GUI) to create the model and run the system.

7.8.1 General Purpose Programming Languages

The use of general purpose programming languages (GPPL) has diminished due to the ease of simulation specific languages or software packages to create, run and extract data from a simulation. The use of a language such as C or C++ could still however suit a particular system to be simulated. The decision to use a GPPL could be affected by the ability of a given programmer and constraints in time, where for example there is inadequate time to learn how a new language or software package operates. Another consideration could be that the level of detail required for a simulation may not be available from any software package currently available.

The programmer must, when using a programming language such as C++, create everything to do with the simulation such as each subroutine that defines how each component of the model acts, to events and entities and how they affect other components. A clock that defines that system at various states and a way to present the gathered data must also be provided (Banks et al. 2001, p104). The availability of simulation libraries such as CSIM however does help programmers. CSIM is a package of commonly used classes and procedures for use with C or C++ (Mesquite Software 2006). Chapter 4 of Discrete-Event System Simulation (Banks et al. 2001) covers the use of C++ and CSIM in detail and provides examples.

7.8.2 General Purpose Simulation Software (GPSS)

Simulation specific software has been around since IBM first released GPSS in the 1960's (Banks et al. 2001, p115). Due to its ease of use and as it was the first program of its type it became a popular simulation package. It was improved by other companies as well as IBM to make it more user friendly (Dictionary of Computer Languages 1998).

The GPSS/H software was introduced by the Wolverine Software Corporation. It is based on the IBM GPSS software with added features to increase its ease of use and has been continually updated to keep it a powerful tool. It is a simulation language that is programmed like most programming languages in the form of text entry; this is unlike some newer simulation tools that use a graphical user interface where placed objects have properties and behaviours associated with them (Crain 1997). The use of a non

LITERARY REVIEW

graphical user interface is so that GPSS/H can remain versatile enough for a wide range of different implementations (Crain 1997).

Despite the ease of use of newer graphical simulation systems GPSS based software is still in use today. This means that it has been in operation for over 40 years. It was designed by Geoffrey Gordon so that it could be used by “non-programmers” and is still advertised as such (Wolverine Software Corporation 2007).

7.8.3 Graphical Simulation Software

The trend of newer simulation software is in the use of a graphical user interface (GUI). This is where objects such as entity generators are placed onto a screen and can have their properties changed. In this way they act as desired by the programmer to model the system under investigation. Examples of such software are Simulink by the MathWorks Inc. and SIMUL8 from the SIMUL8 Corporation (The MathWorks Inc. 1997; SIMUL8 2007).

Advantages of using such a system are the ease of use to construct systems, allowing even novices to build a simple model easily. It also forces the programmer to follow the basic model building process. However it is not always possible for this type of software to be flexible enough to simulate complicated systems accurately. The model may also become quite large visually and may need to be broken down into smaller and smaller subsystems (Crain 1997). Breaking down a system into smaller subsystems can also be seen as an advantage. The programmer can more easily see if any part of the model is not performing as intended when debugging the model.

These types of simulation program usually provide easy and quick methods for displaying results without any previous programming experience or without transferring the data into another program such as Microsoft Excel.

7.9 MATLAB, Simulink and SimEvents

MATLAB is a highly flexible development environment. It allows easy data analysis and visualisation through a high-level technical computing language. It provides an interactive environment for algorithm development and numeric computation (The MathWorks, Inc. 2007a). Add-ons such as Simulink extend the

LITERARY REVIEW

functionality of MATLAB. In this case Simulink provides a new graphical way to implement simulation models. To make implementing various applications easier, additions called ‘Toolboxes’ are supplied. There are a number of toolboxes available for MATLAB such as the fuzzy logic toolbox, neural network toolbox and image processing toolbox to name a few (The MathWorks, Inc. 2007a). Figure 7.9 (The MathWorks, Inc. 2007b) shows the relationship between MATLAB, Simulink and the applications that can be developed using them.

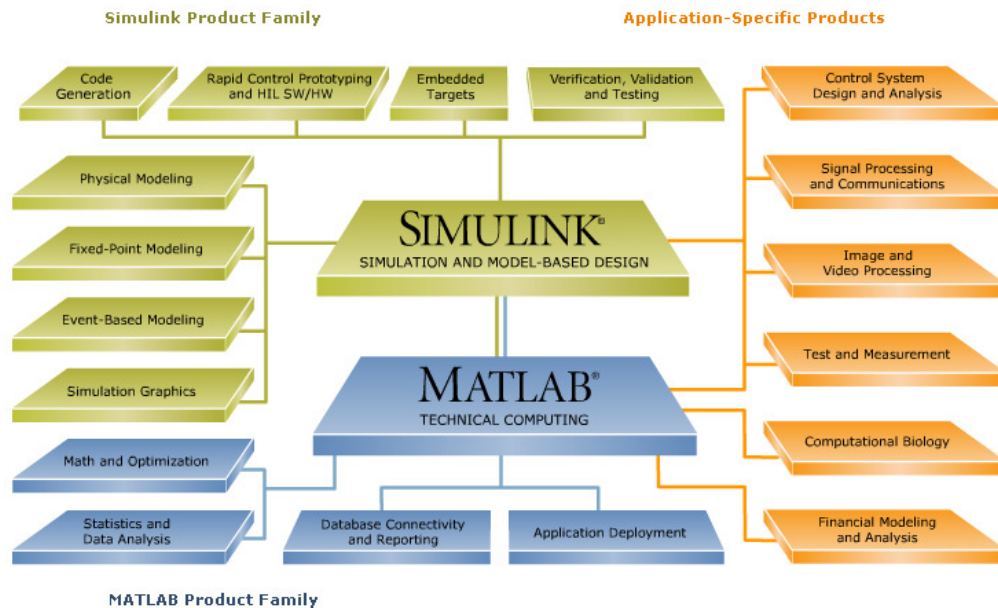


Figure 7.9: MathWorks product overview

7.9.1 MATLAB

Cleve Moler was a math professor at the university of New Mexico where he wanted his students to be able to use computers to solve problems using EISPACK and LINPACK. This required writing Fortran programs however and he didn't want his students to have to learn how to write Fortran programs. In the late 1970s after reading a book by Niklaus Wirth he used Fortran and portions of LINPACK and EISPACK to develop the first version of MATLAB. There was only a matrix data type with 80 functions and to add a function, you had to modify the source code and recompile the program (Mohler 2004). Figure 7.10 (Moler 2004) shows the basic graphical representations from the first MATLAB. Figure 7.11 shows the same result run on a

LITERARY REVIEW

modern version of MATLAB. This shows how the program has developed over the years.



Figure 7.10: First MATLAB graphics

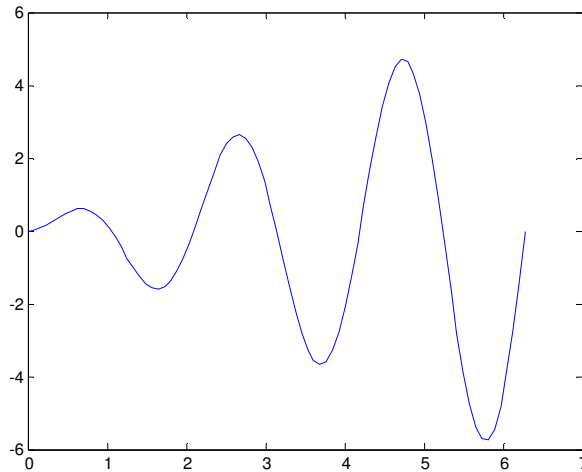


Figure 7.11: Modern MATLAB graph

In 1981 Jack Little, the CEO of The MathWorks realised the importance of the newly released PC from IBM. He and a colleague of his, Steve Bangert, reprogrammed MATLAB in C, adding in extra features and graphical power, and in 1984 The MathWorks Inc. was founded. The founding members were Jack Little, Steve Bangert and Cleve Moler (Moler 2004). Since then The MathWorks Inc. has grown and in 1993 registered one of the first commercial websites and a version of MATLAB to run on the Windows operating system. A Linux version was later released in 1995 (Moler 2006). Figure 7.12 (Moler 2006) again shows the growth of the power and graphical abilities of the software. Figure 7.12 is of an L-shaped membrane which is the company logo.

LITERARY REVIEW

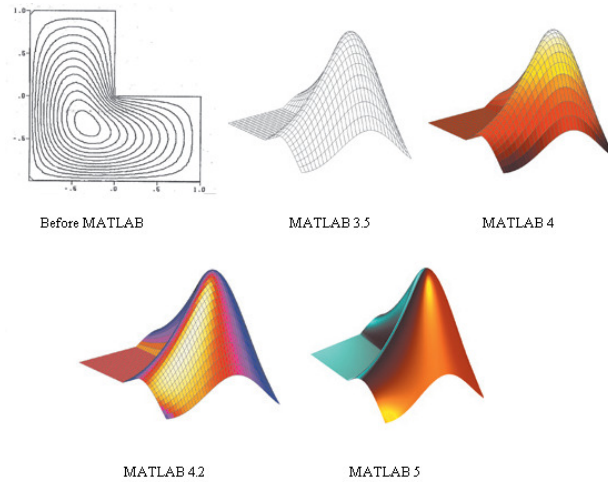


Figure 7.12: MATLAB graphical development

As of the 1st of September 2007 the current version of MATLAB is R2007b. It comes with MATLAB 7.5 as well as Simulink 7 (The MathWorks, Inc. 2007a).

7.9.1.1 MATLAB Development Environment

The main MATLAB window is shown in Figure 7.13; this is the window that opens when MATLAB is initially started. There are three main windows: the current directory you are working from, a command history window and the main command window where code or commands can be entered. These commands can change how a piece of code runs or to run code held in “m-files”.

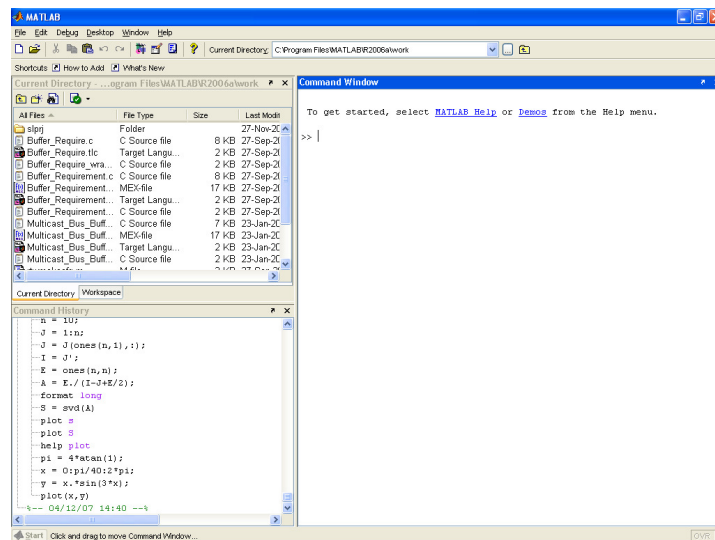
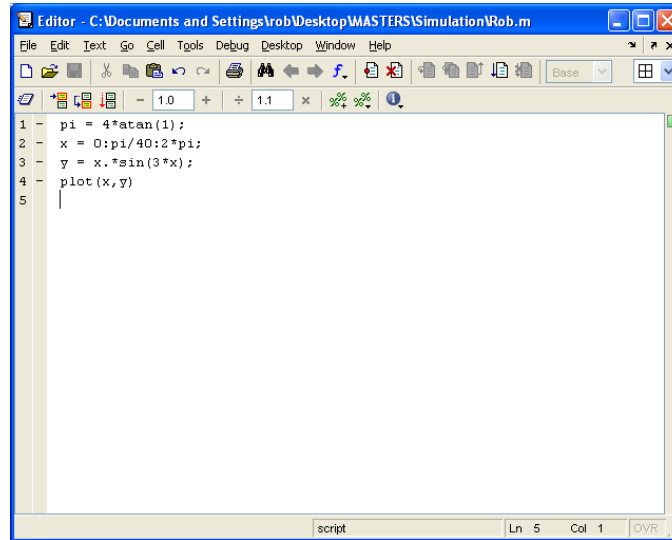


Figure 7.13: MATLAB environment

LITERARY REVIEW

An m-file is a file where code can be entered and stored for retrieval later. It can then be run again and again without having to write the code again and is like a '.c' file for c-code. Figure 7.14 shows an m-file with code to generate the graphs shown in Figures 5.5 & 5.6. This code was taken from Moler (2004).



```
Editor - C:\Documents and Settings\rob\Desktop\MASTERS\Simulation\Rob.m
File Edit Text Go Cell Tools Debug Desktop Window Help
1 - pi = 4*atan(1);
2 - x = 0:pi/40:2*pi;
3 - y = x.*sin(3*x);
4 - plot(x,y)
5 - |
```

Figure 7.14: An m-file

7.9.2 Simulink

Simulink has become widely used by both industry and academic modellers for simulation of dynamic systems since its release in 1990 (The MathWorks, Inc 1999; Moler 2006). As it is embedded in MATLAB it comes with all the analysing ability of MATLAB so that results can be easily displayed, analyzed and interpreted all in the same environment as they were obtained (The MathWorks, Inc 1999). MATLAB can also export its data and graphs easily to other software such as Microsoft Word or Excel.

Where MATLAB stores commands in an m-file and runs commands which look similar to lines of code or mathematic expressions, Simulink is a model building tool where models are built using a GUI. To place an object that forms the model into the model window shown in Figure 7.15 is a simple case of 'dragging and dropping' a virtual object. Figure 7.15 also shows a simple model that was built using Simulink. Figure 7.16 is the Simulink library where the objects are selected. There is also the ability to make your own blocks.

LITERARY REVIEW

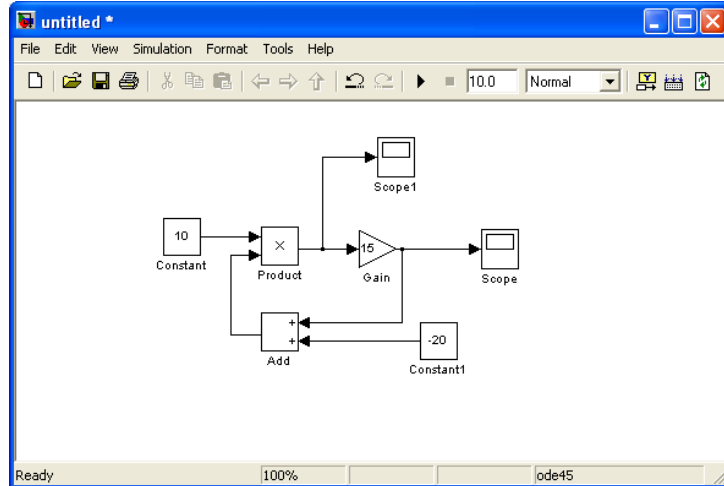


Figure 7.15: Simulink environment

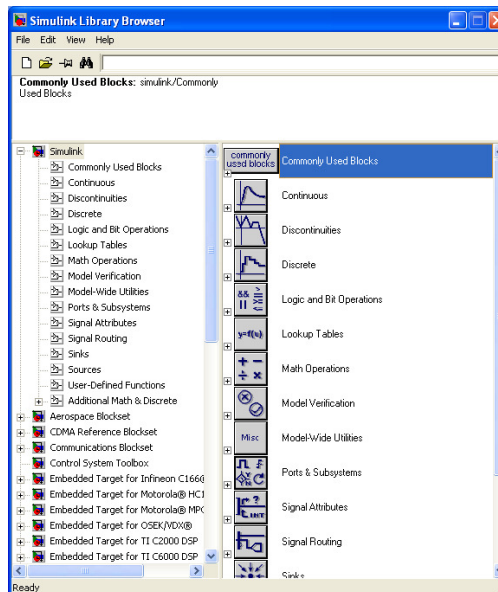


Figure 7.16: Simulink library

The models that can be built can be 'hierarchical' so that they are easier to visualise. Each level can then be entered into to view the level below. The models can however be built from the top down or the bottom up. This allows a developer to approach the model building in a way that helps them understand the function of the system (The MathWorks 1999).

Once the model is built it can be run and data easily obtained. The parameters of the system can then be changed quickly and the test run again to see how these changes

affect the performance of the system and these results easily compared to the previous results (The MathWorks 1999).

7.9.3 SimEvents

SimEvents is an add-on to Simulink. It allows for discrete-event simulation to be achieved through the additional components that come as part of the package. Activity-based models are created by the modeller to evaluate system parameters. Entities are configured with attributes which can then be used to model applications such as packet-based networks, real-time operating systems and computer architectures (The MathWorks, Inc. 2007a).

As was stated SimEvents has its own components associated with it. These can be found in the Simulink library under the heading SimEvents. The SimEvents library can also be opened in its own window as shown below in Figure 7.17.



Figure 7.17: SimEvents library window

When one of the boxes shown in Figure 7.17 is opened it will produce another set of options; these could be options that refine the selection you made, i.e. when selecting the generators block you get another set of options. In this case it would be event, entity or signal generators. When there are no more variations to pick from then a selection of objects to choose from will be displayed. These can then be dragged and dropped onto the model screen. Figure 7.18 shows the two different types of entity generator that can be selected.

LITERARY REVIEW

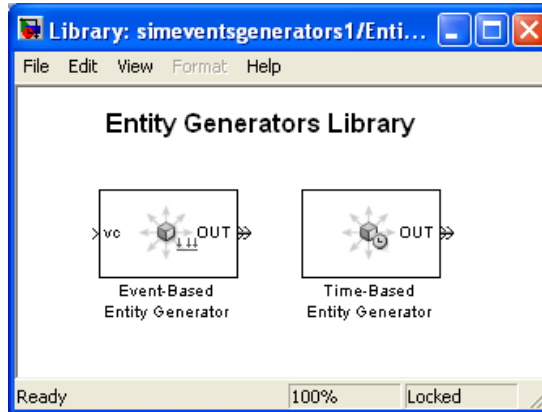


Figure 7.18: SimEvents entity generator objects

SimEvents contains all basic blocks as standard to create a model for a discrete-event simulation. With MATLAB and Simulink it is a flexible program that can be used to model many systems.

7.9.4 SimEvents Model Building Tutorial

This section is based on the 'Building a Simple Discrete-Event Model' section of the SimEvents – Getting Started Guide (The MathWorks, Inc. 2007c). All diagrams are also taken from the SimEvents – Getting started Guide.

The system is a simple queuing system in which entities arrive in a deterministic way, to a queue, and proceed to a server that operates at a fixed rate. The type of system is a D/D/1 queuing system which implies a deterministic arrival rate, a deterministic service rate, and a single server.

As was previously stated, SimEvents provides a library of simulation blocks. These include all necessary blocks to perform discrete event simulation such as a server block and an entity generator block. Other blocks include blocks to both set and read attributes as well as versions of Simulink blocks to be used with SimEvents. One such block is the SimEvents Signal block. It is important to use any special SimEvents blocks where possible. Results obtained may represent false information if Simulink blocks are used.

To build the simple D/D/1 queuing system for this example the following types of blocks should be chosen:

LITERARY REVIEW

- An entity generator is needed. The time based entity generator will be used for this example.
- A queue is needed and the FIFO queue was used in this example.
- A server is also needed. As it is a single server structure, the single server option was chosen.
- An entity sink was also needed. This was to accept any entity after it passes through the server.
- Finally a SimEvents signal scope is needed to display data.

It is important to choose the correct block to accurately reflect the system being modelled. In most cases there is a choice of what block is needed to model the system. If the queuing system is a LIFO then choosing the FIFO queue would cause the model to act in an incorrect way. The correct blocks used in this example are shown in Figure 7.19.

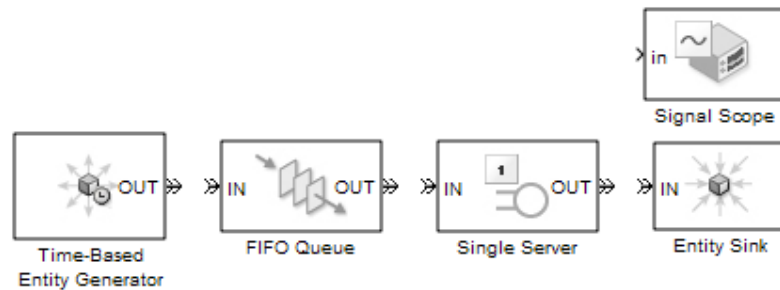


Figure 7.19: SimEvents tutorial blocks

Once the correct blocks are chosen, they must be configured. For example the time-based entity generator needs to know how often to generate an entity. In this example the entity generator was setup to produce an entity every second. The generator was also set to generate an entity at the start of the simulation. The service time of the server was also set as 1. These settings are all done through the parameters dialog box of the appropriate blocks.

However to obtain any data the scope must be able to obtain some information from the system. The single server's parameters block is shown in Figure 7.20. Figure 7.19 shows that the 'number of entities departed' option was set to 'on' for the single server.

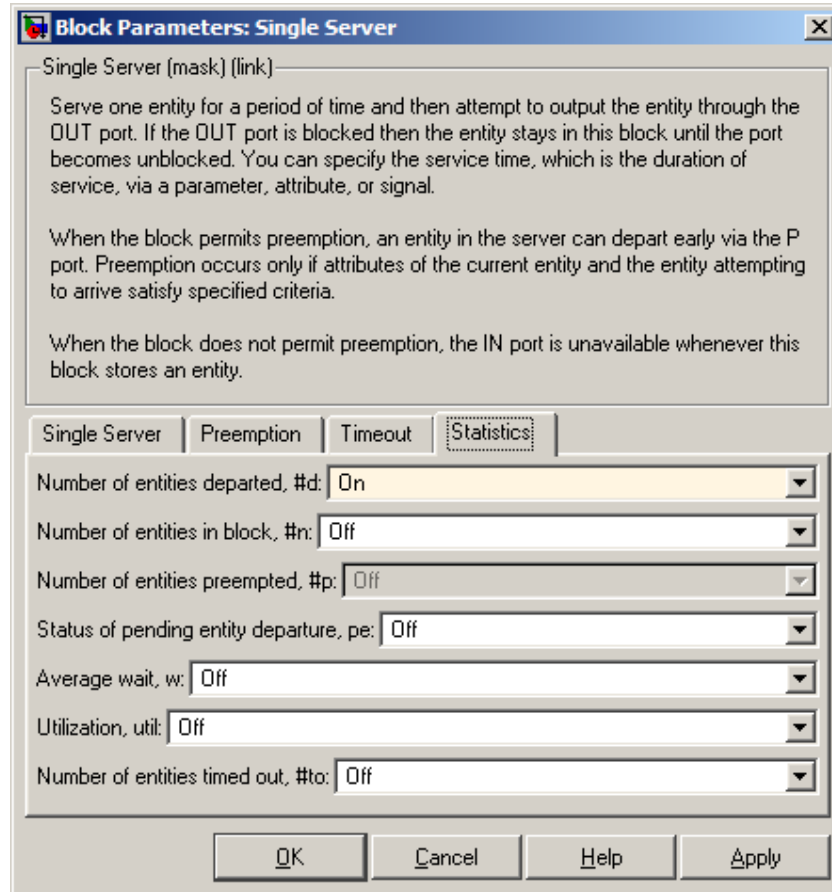


Figure 7.20: Single server parameters box

When a statistics option is enabled, a new output port is added to the block. This can then be connected to the input of the signal scope. A path for entities must also be made between the different blocks. The correct path configuration is shown in Figure 7.21. Note the new output on the signal server marked as '#d'. The different connection types do not allow a signal port to be connected to an entity path port and vice versa. Also an output port cannot be connected to another output port, while inputs ports cannot be connected to other inputs ports.

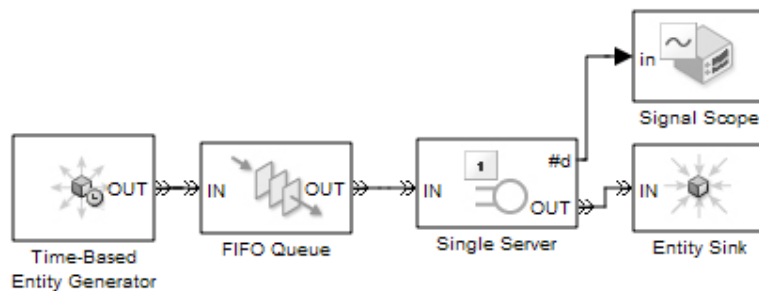


Figure 7.21: Tutorial blocks connected

LITERARY REVIEW

When the model is built the simulation can then be run. Figure 7.22 shows the results displayed on the signal scope after the model was run for 10 seconds. Other results would be obtained if different statistics were monitored, or if different parameters were used for the service time or arrival rates.

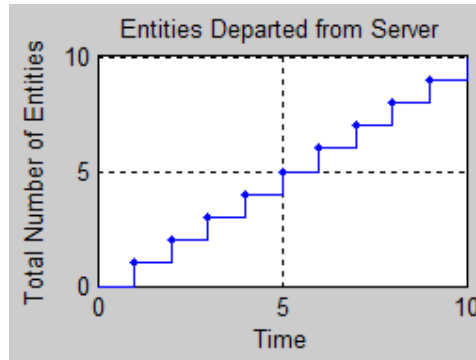


Figure 7.22: Tutorial results

Large systems can be constructed by following the same steps as described above. Small subsections can be created and tested one by one and added to an overall system model. Each subsystem needs only to reflect the behaviour of the basic operation of that subsystem. The systems parameters such as service time can then be configured and simulations run.

7.10 Simulation Software Selection

Shannon (1975, pp107-109) and Banks et. al. (2001, pp100-103) both propose steps for choosing simulation software. The steps and questions proposed can be combined as follows:

Part1: Investigation of Possibilities and Initial Screening

1. An important factor in choosing the software is hardware availability. If a new computer is to be purchased it can be configured to suit the needs of the modeller. In most cases this is not possible. It is therefore important to know the limits of the hardware.

LITERARY REVIEW

2. Is there sufficient documentation or vendor support available? If this is not the case then problems encountered may not be easily dealt with. Alternatives could however be internet forums where help can be obtained from regular users of the software.
3. Are the advertising claims accurate? If there is a checklist with 'yes' and 'no' entries does the variation or licence you are interested in purchasing provide all the required features. Do the implementation and capabilities of the features match what you are looking for?
4. Is the package able to generate a model that is cross platform compliant? A cross platform compliant model has wider use than if it is only executable on a single operating system.
5. The speed of the simulation should be considered. Debugging of the model may take a long time if the model is slow to execute.
6. The costs of software packages can be very high. What is the range of software packages available for your budget?
7. The ease of which the simulation package can be learnt. A graphical package may remove the need to learn syntax but it will not remove the need to use procedural logic.

Part 2: Overall Choice Based on Problems to be Solved

1. The type of simulation to be run is evaluated. Based on this an appropriate software package can be chosen. Questions to ask should include: Is the model an event-, process- or activity-oriented system? Is there a reliance on random numbers? If so what is the capability of the software to generate the random numbers?
2. Data needs to be obtained for analysis therefore the ease of obtaining and storing data, i.e. format, needs to be investigated
3. The software should be flexible enough to meet your needs. Can functions written in a general purpose software language be included if needed? If so what programming languages does it support?

As can be seen, based on the considerations proposed above a suitable software package can be chosen. The selection process is broken down into two parts. The first

LITERARY REVIEW

part is intended to discard software that may be more difficult to use, adapt or costly. The second part tries to identify, from the remaining choices, the most suitable software package for the problems at hand. In many cases a software package may be quickly marked as unsuitable.

With little or no modification these steps can be applied to any other necessary hardware or software choices that need to be made.

7.10.1 Simulation Software Selection

Table 7.1 shows the options that were considered for the software used to carry out the research as outlined in this thesis. The features and benefits are compared for each possible software package also. Each of the considerations were given a score based on factors such as budgetary considerations and prior knowledge of the package. Each score was rated between 0 and 10, where 10 is the highest score possible for any category and 50 the highest over all score possible.

	Simulation Methods provided	Suitability [†]	Ease of Use	Support	Cost [‡]	Total
C++	0	3	8	6	6	24
GPSS	8	9	7	8	5	36
SimEvents	9	9	8	9	10	45

Table 7.1: Simulation software selection analysis

The most suitable software package based on the comparisons shown in Table 7.1 was deemed to be the SimEvents software add-on package for MATLAB. The MATLAB aspect of the software package made it an attractive option. MATLAB provides an environment where computations on outputs obtained from a model can be analysed. This can then be easily converted into graphical representations (i.e. charts and graphs) if desired. Other benefits also included the fact that the system had been

[†] The suitability metric was calculated based on the information as covered in section 7.8 and the requirements to carryout the research correctly.

[‡] This was based on the price for a student version, with MATLAB with SimEvents priced around €115. The Wolverine GPSS/H cheapest price was €273 and a copy of MicroSoft Visual Studion 2008 standard edition was price at €234. All prices are approximate values.

LITERARY REVIEW

designed to perform discrete event simulation. SimEvents therefore provides a number of features, such as a simulation clock, as standard. While generic programming languages could have been used, features such as the simulation clock also need to be created. However a generic simulation package could be have been chosen. This would have had the benefit of not requiring the developer to learn a new programming language. The graphical method of creating a model in SimEvents was deemed easy to learn. Therefore minimum time would be spent learning a new software package. A big factor was also the cost of any software package used. The MATLAB and SimEvents price was the cheapest software package. The price also included MATLAB which can be used to analyse data.

The support provided for SimEvents is the best of all three packages looked at. The help files are not only extensive, but the online help and user forms have a large number of people who participate in them. SimEvents therefore also proved to be the best value for money. This allowed the best value combination of hardware and software to be obtained to conduct the research.

It should be noted that no one option is necessarily better for the purpose of conducting a similar research topic. Other studies may prefer to use a GPPL as it would provide an easier way to streamline the model in terms of execution speed and memory demands. For the constraints on the research conducted in this thesis SimEvents was the most suitable. The scores obtained were biased based on factors such as budget and the developer's exposure to different software packages and this is acknowledged.

7.11 Conclusion

Model building has a wide range of uses. These vary from a simple child's toy to a sophisticated model used in simulation of a system for analysis. This simulation again has a wide range of uses, be it the simulation of computer architectures or that of a queue of people waiting in line. These simulations can be run to give the modeller a better understanding of how a system works. They can also be modified to see how a change to a system affects the performance of the system. They can be important as an evaluation tool for a developer of a system to determine how effective his real system is. This is important as it may not always possible to change the real world system in

LITERARY REVIEW

various ways to get feedback. It is also important to consider different simulation possibilities, such as HILS testing, before deciding on using a simulation model. Different approaches to evaluation of system will have different benefits and drawbacks.

For an in-vehicle network such as FlexRay, simulation allows the designer/analyst to see how the system works, and through their greater understanding of the system they can draw conclusions on how to improve the system. This could be in terms of looking at message flow and suggesting for example new levels of RAM, which could have an important impact on the cost of the network as memory is an expensive resource. The scheduling of the network could also be analysed in an easier fashion than taking readings of a real network and improvements could also be drawn from this type of simulation. This makes simulation an important tool in developing technologies as it has a wide range of applications.

The best software and hardware tools can also be chosen using the steps outlined in this chapter. This can have a big impact on the performance and functionality of the model. This means that sufficient thought and time should be put to this process. The software tools available were evaluated against each other and the most suitable software was found to be SimEvents.

7.11. References

Applied Dynamics International (2007) Why use Hardware-in-the-Loop Simulation? [online], available: http://www.adi.com/products_sim_qhilWhy.htm [accessed 2 September 2009].

Banks, J., Carson, J. S., Nelson, B. L. and Nicol, D. M. (2001) Discrete-Event System Simulation, New Jersey: Prentice Hall.

Carter, S. (1996) Markov Chains [online], available at: <http://www.taygeta.com/rwalks/node7.html> [accessed 12 December 2007].

LITERARY REVIEW

Crain, R. C. (1997) Simulation Using GPSS/H, in Andradóttir, S., Healy, K. J., Withers, D. H. and Nelson, B.L. eds., Winter Simulation conference, Atlanta, Georgia, 7-10 Dec, IEEE, 567- 573.

Dictionary of Programming Languages (1998) GPSS [online], available at:
http://cgibin.erols.com/ziring/cgi-bin/cep/cep.pl?_key=GPSS [accessed 3 December 2007].

Didkovsky, N. (1996) What's a Markov Process? [online], available at:
<http://www.doctornerve.org/nerve/pages/interact/markhelp.htm> [accessed 5 December 2007].

Gomez, M. (2001) Hardware-in-the-Loop Simulation [online], available:
<http://www.embedded.com/15201692> [accessed 2 September 2009].

Hwang, T., Roh, J., Park, K., Hwang, J. Lee, K.H., Lee, K., Lee, S. and Kim, Y. (2006) Development of HILS Systems for Active Brake Control Systems, Proceedings of SICE-ICASE International Joint Conference, Bexco, Busan, Korea, October 18-21, IEEE Computer Society Washington, DC, USA, 4404-4408.

Law, A. M. and Kelton, W. D. (1982) Simulation Modeling and Analysis, New York: McGraw-Hill.

Mitrani, I. (1982) Simulation Techniques for Discrete Event Systems, Cambridge: Cambridge University Press.

Mohler, C. (2004) MATLAB News & Notes – December 2004, Cleve's Corner – The Origins of MATLAB [online], available at:
http://www.mathworks.com/company/newsletters/news_notes/clevescorner/dec04.html
[accessed 4 December 2004].

Mohler, C. (2006) MATLAB News & Notes – January 2006, Cleve's Corner – The Growth of MATLAB and The MathWorks Over Two Decades [online], available at:

LITERARY REVIEW

http://www.mathworks.com/company/newsletters/news_notes/clevescorner/jan06.pdf

[accessed 4 December 2004].

Neelamkavil, F. (1987) *Computer Simulation and Modelling*, New York: Wiley.

Ripley, B.D. (1987) *Stochastic Simulation*, New York: Wiley.

Shannon, R.E. (1975) *Systems Simulation – the Art and Science*, New Jersey: Prentice Hall.

SIMUL8 Corporation (2007) [online], available at:

<http://www.simul8.com/products/standard/index.htm> [accessed 3 December 2007].

Slater, T. (2000) *Queuing Networks – Network 2* [online], available at:

<http://www.dcs.ed.ac.uk/home/jeh/Simjava/queueing/Networks/networks.html>

[accessed 12 December 2007].

The MathWorks, Inc. (1999) *Using Simulink version 3*, Massachusetts: The MathWorks, Inc.

The MathWorks, Inc. (2007a) *The MathWorks - MATLAB and Simulink for Technical Computing* [online], available at <http://www.mathworks.com/index.html?ref=pt> [accessed 3 December 2007].

The MathWorks, Inc. (2007b) *MathWorks Product Overview* [online image], available at <http://www.mathworks.com/products/pfo/> [accessed 4 December 2007].

The MathWorks, Inc. (2007c) *Getting Started with SimEvents 2*, Massachusetts: The MathWorks, Inc.

The MathWorks, Inc. (2007d) *SimEvents 2 User's Guide*, Massachusetts: The MathWorks, Inc.

LITERARY REVIEW

Weisstein, E. W. (2007a) Markov Process [online], available at:
<http://mathworld.wolfram.com/MarkovProcess.html> [accessed 5 December 2007].

Weisstein, E. W. (2007b) Monte Carlo Method [online], available at:
<http://mathworld.wolfram.com/MonteCarloMethod.html> [accessed 7 December 2007].

Willig, A. (1999) A Short Introduction to Queuing Theory [online], available at:
<http://www.tkn.tu-berlin.de/curricula/ws0203/ue-kn/qt.pdf> [accessed 12 December 2007].

Woller, J. (1996) An introduction to Monte Carlo Simulations [online], available at:
<http://www.chem.unl.edu/zeng/joy/mclab/mcintro.html> [accessed 7 December 2007].

Wolverine Software Corporation (2007) GPSS/H - Serving the simulation community since 1977 [online], available at: <http://www.wolverinesoftware.com/> [accessed 3 Dec 2007].

Wolfson, M. M. and Pert, G.J. (1999) An Introduction to Computer Simulation, Oxford: Oxford University Press.

Xun, W., Shu-xing, Y. and Lei Z. (2008) Dynamic Test and Evaluating System for Flight Control System, Proceedings of the 2008 International Colloquium on Computing, Communication, Control, and Management, Guangzhou City, China, August 3-4, IEEE Computer Society Washington, DC, USA, 189-192.

Zhu, Y., Hu, H., Xu, G. and Zhao, Z. (2009) Hardware-in-the-Loop Simulation of Pure Electric Vehicle Control System, International Asian Conference on Informatics in Control, Automation and Robotics, Bangkok, THAILAND, February 1-2, IEEE Computer Society Washington, DC, USA, 254-258.

Chapter 8 . FlexRay Software Drivers

8.1 Introduction

The research described in this thesis concerns improving the flow of data through a FlexRay node. An important area that should be looked at is the software driver linking the communication controller and the host. How data is handled at this stage could have an enormous effect on the performance of the node. Data is passed from the host to the communications controller for transmission. If the data is not passed in adequate time, it may not transmit during its allocated slot time. It will then have to wait until the next communication cycle before the data is transmitted. It is therefore necessary to know how the driver performs to fully optimise the node.

Software drivers are intended to provide an abstract interface between hardware and user defined pieces of software (Dependable Computer Systems 2006). This chapter will outline the DECOMSYS::COMMSTACK FlexRay software driver. It will introduce AUTOSAR and the method it uses to transfer data between software components. The Fujitsu FlexRay driver will also be covered. Figure 8.1 shows the various software driver options available to a FlexRay based system. A system designer must choose one of the software options for the implementation on the host microcontroller. In Figure 8.1 the options are (a) AUTOSAR FlexRay stack, (b) the DECOMSYS::COMMSTACK FlexRay software driver and (c) the Fujitsu FlexRay driver.

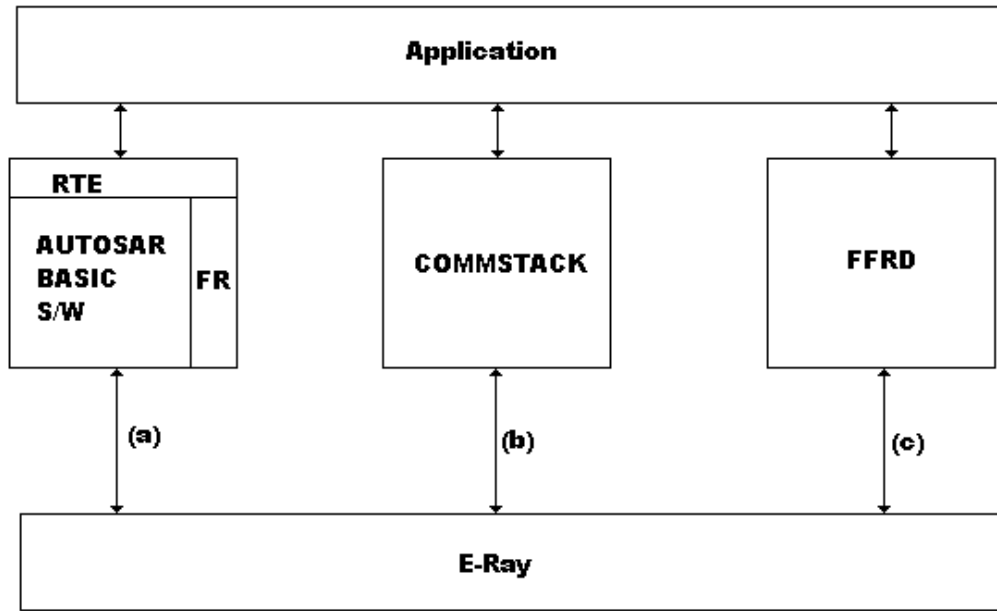


Figure 8.1: FlexRay software driver options

8.2 COMMSTACK

The DECOMSYS::COMMSTACK<FlexRay> is a software driver (here after referred to as ‘driver’) designed to provide a FlexRay interface to specific hardware implementations. It was designed with flexibility in mind. As such it is not dependant on other external components for operation (Dependable Computer Systems 2006).

DECOMSYS::COMMSTACK<FlexRay> allows higher layer software to be developed with little sense of the behaviour and properties of a FlexRay node. This allows the designer to be unaware of the actual communication controller implementation (Dependable Computer Systems 2006). Figure 8.2 (Dependable Computer Systems 2006, p4), shows a system implementing the (here after referred to as ‘COMMSTACK’).

LITERARY REVIEW

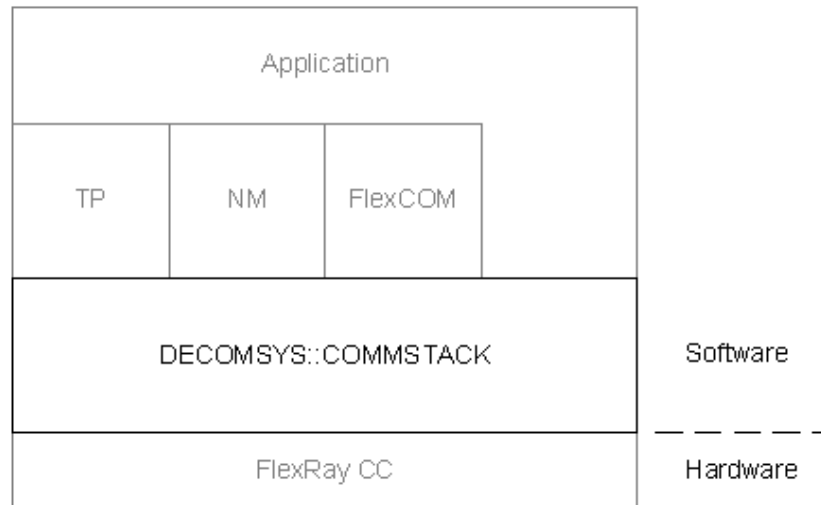


Figure 8.2: COMMSTACK system overview

`DECOMSYS::COMMSTACK<CONFIGURATOR>` is a plug-in for DECOMSYS Designer. It outputs the following files (Dependable Computer Systems 2004, p5):

- A '.h' file containing basic configuration options
- A '.c' file containing frame identifier and queue configurations
- A '.c' file implementing the host specific communication controller initialisation functions.

8.2.1 System Design

The COMMSTACK internal structure can be seen in Figure 8.3 (Dependable Computer Systems 2006, p6). It is broken down as follows (Dependable Computer Systems 2006, p7):

- **FlexRay Hardware:** One of several communications controllers will be used for an application.
- **Hardware Configuration:** This contains hardware mapping information.
- **Application Configuration:** A post-build configuration holds all application specific configurations.
- **COMMSTACK:** This is ported to the dedicated host-CPU for a specific development environment.

LITERARY REVIEW

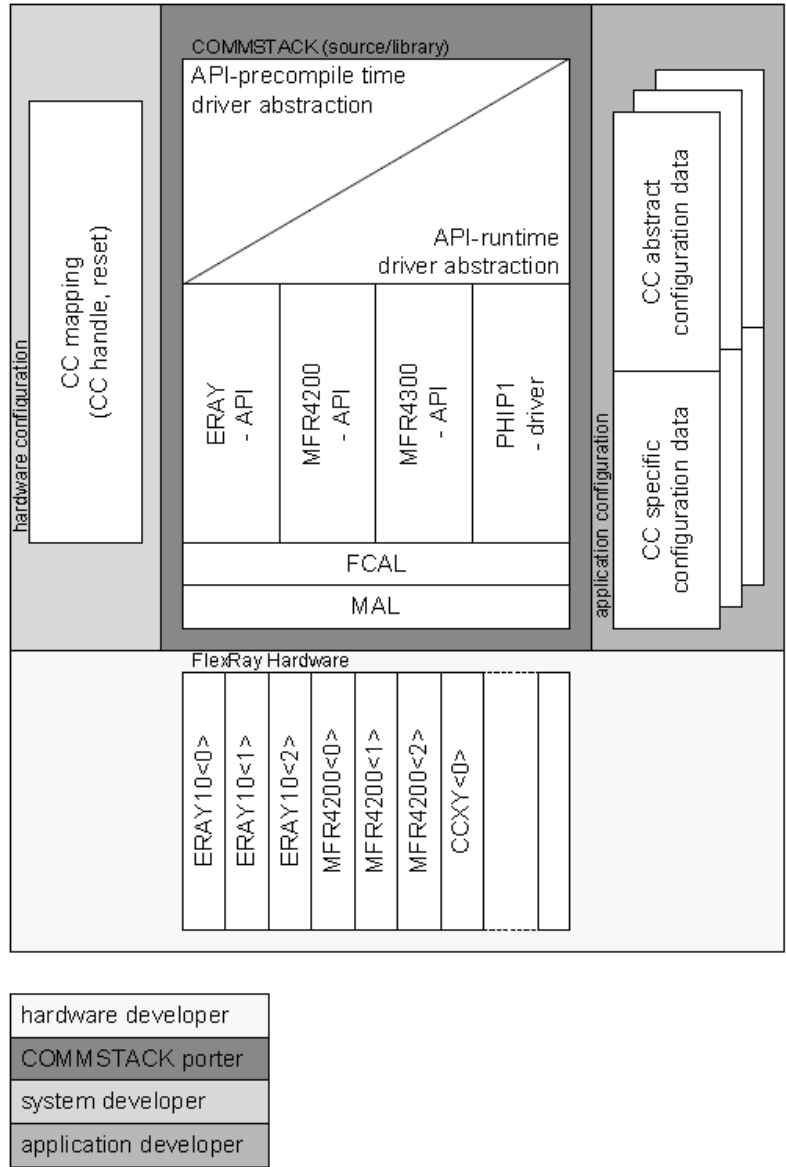


Figure 8.3: COMMSTACK system architecture

8.2.1 System Design

The behaviour of COMMSTACK is defined by a state machine. This can be seen in Figure 8.4 (Dependable Computer Systems 2006, p7). For further information on each state see Dependable Computer Systems (2006, pp8-10).

LITERARY REVIEW

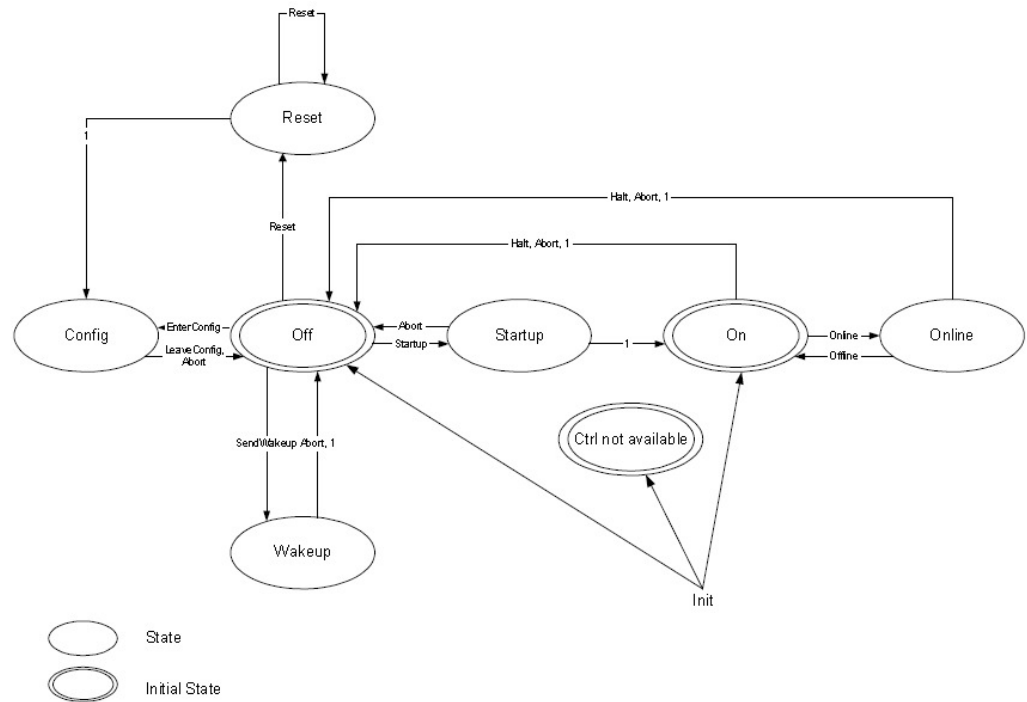


Figure 8.4: COMMSTACK state diagram

8.3 AUTOSAR

AUTOSAR was developed from 2003 when the core partners of the AUTOSAR partnership signed a contract. Since then the AUTOSAR partnership has developed a number of specifications (Fennel et. al. 2006). These companies worked together to produce the AUTOSAR standard to support automotive electronic/electrical developments to meet current and future needs of the automotive industry (AUTOSAR GbR 2008).

A major focus of the AUTOSAR partnership is the re-use of software components. Traditionally software components were developed with a hardware focus. This approach leads to difficulty when replacing or upgrading hardware components. By developing a run time environment to support re-use of software components a reduction in costs and complexity can be achieved.

8.3.1 AUTOSAR Goals

As was stated one of the main focuses of the AUTOSAR partnership is the development of the necessary tools and specifications to achieve re-use of software components. To realise this the main goals initially were to establish a standard for use in future vehicle applications (AUTOSAR GbR 2008a). In the first phase of AUTOSAR the focus was on powertrain, chassis, active and passive safety, body and comfort. Applied to these the main objectives were a consideration of the following desirable elements (AUTOSAR GbR 2008a):

- Consideration of safety requirements.
- Scalability for different platforms and vehicles.
- Standardisation of basic system functions.
- The ability to move functions to different nodes on the network.
- Integration of modules from different suppliers.
- Maintainability.
- Increased use of ‘off the shelf’ hardware.
- Software updates/upgrades for vehicles.

8.3.2 Virtual Functional Bus

To achieve their goals and objectives the Virtual Functional Bus (VFB) concept was developed by the AUTOSAR partnership. Using a VFB AUTOSAR is able to separate the functionality of a node into the software components (applications) and the basic software (communications methods for example) of the hardware module. The communication is handled by the basic software on a node. The applications functionality is obtained from combinations of software components. The application layer is then executed using a Run Time Environment (RTE). Figure 8.5 (AUTOSAR GbR 2008b, p10) shows how the decoupling of the software is achieved.

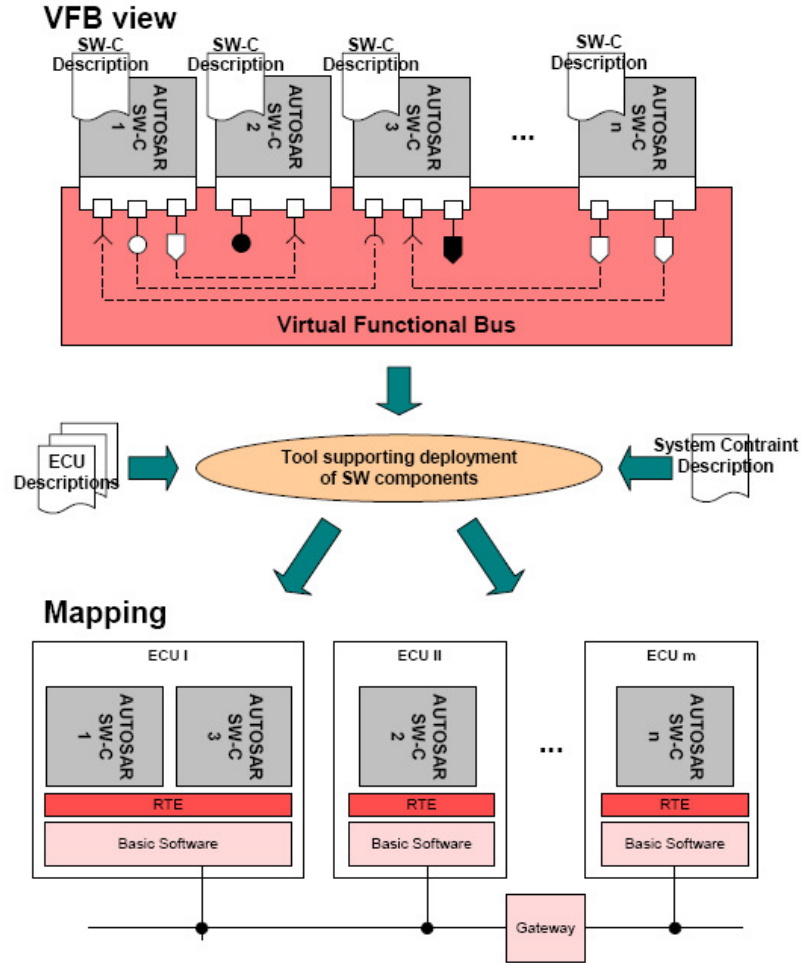


Figure 8.5: Virtual functional bus concept

Data to be sent from one software component to another is conceptually sent through the VFB during the development process. At compile time the software components are then mapped to specific hardware modules.

The programmer will conceptually send and receive information from other software components through the VFB. At compile time however the VFB is replaced by an application programming interface (API). This means that if one software component needs to pass data to another software component or request a service supplied by another software component, it does so through the use of API calls. So at compile time only the functionality necessary to achieve this is set up in the basic software portion of the node. This eliminates unnecessary code generation for the module. Another advantage is that the software component no longer needs to know

where the other software component is located. It only needs to know what services or information it can accept or provide.

8.3.3 AUTOSAR Components and Interfaces

There are three types of components defined by AUTOSAR (Buttle 2005):

1. Atomic software components
2. Sensor/actuator software components
3. Composite components

A software component is assigned to one of these categories based on its functionality. Each software component can have as many interface ports as needed.

The interface types are broken down into:

1. Provided interfaces
2. Required interfaces

The communication is then defined as either a sender-receiver or a client-server type. Sender-receiver is like a publish-subscribe type interface where information is sent out and anybody can take that information. Client-Server can be seen as a function calling interface (Jackman 2008).

Figure 8.6, shows the different type of component interface. Figure 8.5 is based on a diagram by Buttle (2005).

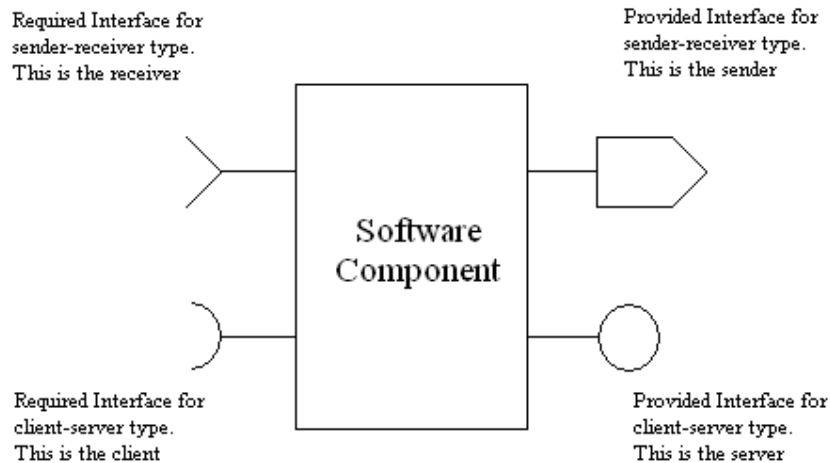


Figure 8.6: Software component communication interface types

8.3.4 AUTOSAR FlexRay Stack

Figure 8.7 (TTTech Automotive GmbH 2007) shows where the FlexRay stack components mapped onto the basic software of a hardware module using AUTOSAR.

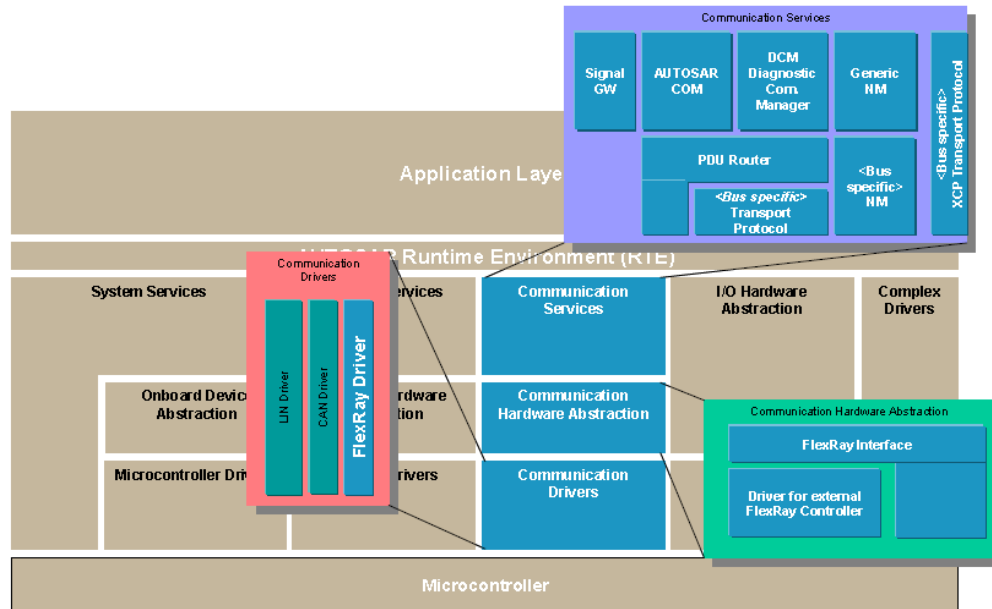


Figure 8.7: FlexRay stack layout

The AUTOSAR FlexRay stack consists of a number of components and layers. These components and layers include a FlexRay driver. It is designed to allow a software developer to know little or no information about the underlying FlexRay system. The programmer merely passes information from the application and the FlexRay stack is designed to handle the information. In this way the programmer need only know how to use a hardware independent API call.

The FlexRay stack is made up of: FlexRay-specific modules, drivers, the interface layer, the protocol data unit (PDU) router, the FlexRay transport protocol (TP) layer and the protocol-independent communication (COM) layer. In some cases a communication-specific layers like the network management (NM) layer also defined. The FlexRay driver is part of a microcontroller abstraction layer that provides access to the FlexRay controller through a hardware-independent API (Weka Fachmedien GmbH 2008).

LITERARY REVIEW

The following definitions of the different FlexRay stack components are from Galla et. al. (2007):

- The transport protocol module is used to segment and reassemble large PDUs. A PDU is simply a message or frame of a particular networking scheme. The PDUs are transmitted from and to the Diagnostic Communication Manager.
- The PDU router is used to either send messages to higher protocol layers or to perform a gateway service. This could mean gating the message between two FlexRay networks or between FlexRay and another networking scheme such as CAN.
- The COM module provides signal-based communication to the run-time environment. This can be in the form of inter-ECU or intra-ECU communication.
- The Diagnostic Communication Manager provides a way to allow tester devices to control diagnostic functions in an ECU using the communication network.
- The network management module provides a coordinating mechanism for the ECUs on the network. It is split between a generic network management and a protocol specific network management scheme.
- The FlexRay interface module facilitates the transmission and reception of the PDUs. It allows multiple PDUs to be packed into a single frame at the transmission ECU and to be successfully extracted again at the receiving ECU. This is affected by the timing constraints of the FlexRay protocol. The packed PDUs are sent to and received from the FlexRay driver.
- The FlexRay driver provides the basis for the FlexRay interface module by facilitating the transmission and reception of frames to and from a communication controller. It too is affected by the timing constraints of the FlexRay protocol.

8.4 Fujitsu FlexRay Driver

The Fujitsu FlexRay driver is intended to ease the familiarisation phase of using FlexRay for developers (Fujitsu Microelectronics Europe 2007, p8). It supports the 32-bit MB91460 family processors, MB88121 series communication controllers, 16-bit MB96340 family processors and the MB91F465X series processors with integrated

communication controllers. It is designed to be compatible with DEYCOMSYS::DESIGNER (Fujitsu Microelectronics Europe 2007, pp9-10).

8.4.1 Driver Concept

The driver is designed to be viewed as several layers. The minimal number of layers and their function can be seen in Figure 8.8 (Fujitsu Microelectronics Europe 2007, p11).

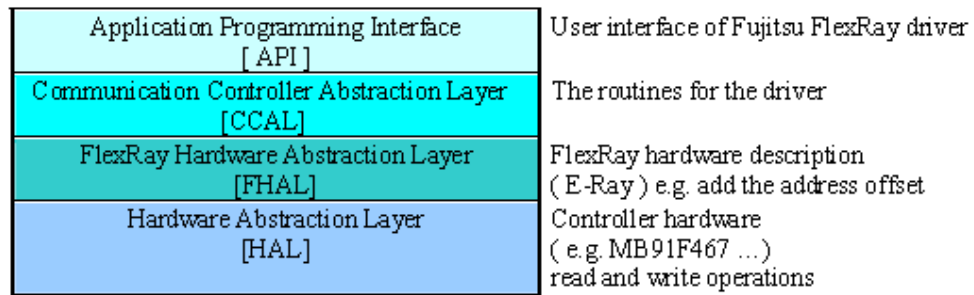


Figure 8.8: Fujitsu FlexRay driver layers

The Architecture of the driver is shown in Figure 8.9 (Fujitsu Microelectronics Europe 2007, p11).

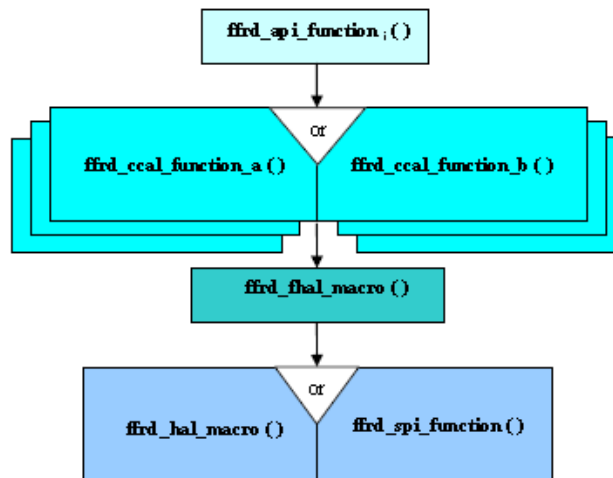


Figure 8.9: Fujitsu FlexRay driver architecture

LITERARY REVIEW

The `ffrd_api_function()` evaluates one of 90 API calls available for the FlexRay driver. It then calls a relevant routine from `ffrd_ccal_function()`. This layer contains all the routines to handle the API calls. Following this the macro from `ffrd_fhal_function()` is called to add the offset address for the E-Ray chip. Using `ffrd_hal_function()` the macros for different MCU-FlexRay controller access is located. It should be noted that the files, macros and functions are only included at compile time if needed (Fujitsu Microelectronics Europe 2007, p12).

8.4.2 Program Flow

The services of the FlexRay driver is shown in Figure 8.10 (Fujitsu Microelectronics Europe 2007, p15). As can be seen, the initialisation service and some of the control services and status services are available after a reset. When the initialisation service is completed all other services are available (Fujitsu Microelectronics Europe 2007, p15).

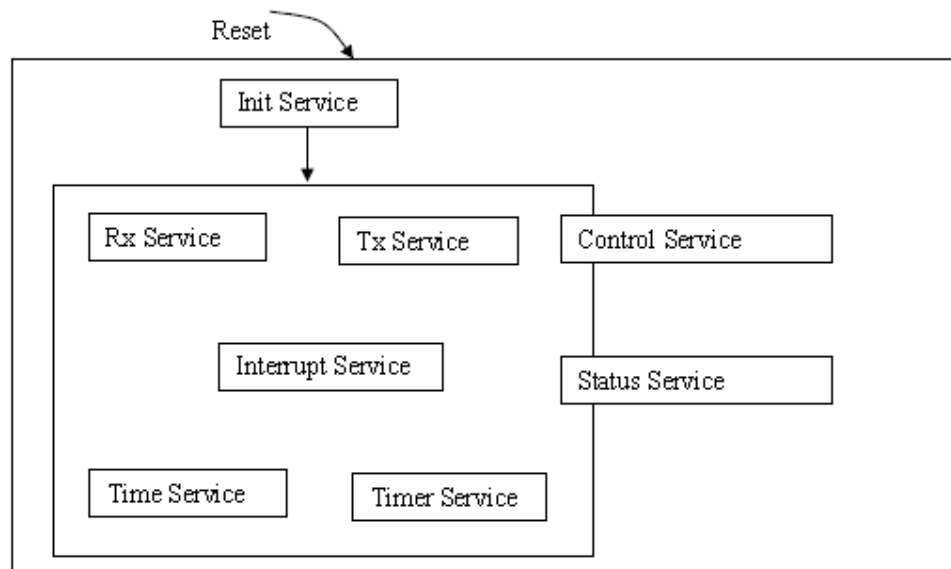


Figure 8.10: Fujitsu FlexRay driver services

8.5 Conclusion

It is important to know how the data is transferred from the host to the communications controller. If data is not transferred in a timely fashion it may miss the transmission slot it is assigned to. If this is the case the data will not be sent out during the current communication cycle. This may lead to problems as the data will be used for decision making or in calculations. The FlexRay drivers outlined above are designed to be used for a wide range of systems.

AUTOSAR looks set to be adopted by a large section of the automotive industry. The driver could therefore have a huge effect on the performance of networks. COMMSTACK is a well defined library of necessary FlexRay interface functions. This makes it a very useful driver for non-AUTOSAR applications. The Fujitsu FlexRay driver is a useful tool to allow early development and familiarisation. It can also be used for lower level functions which could be useful for some applications, i.e. timing analysis.

Both the COMMSTACK and FFRD software drivers will be used in this research. COMMSTACK is the software driver that shall be modelled. The features and timing of this will be implemented into the simulation model. The FFRD driver will be used in the calibration and validation stages of the model building process. This is used to obtain timing information from the real world system implementations. Zhu (2007) already developed a simulation of an AUTOSAR based system and so will not be used in this research.

8.6 References

AUTOSAR GbR (2008a) AUTOSAR [online] available at: www.autosar.org [accessed 15 May 2008].

AUTOSAR GbR (2008b) Specification of the Virtual Functional Bus version 1.0.1, Munich, Germany.

Buttle, D. (2005) What is an RTE?- Introduction to AUTOSAR for RTE users, Stuttgart, Germany.

LITERARY REVIEW

Dependable Computer Systems (2004) DECOMSYS::COMMSTACK
<CONFIGURATOR> User Manual, Wein, Austria.

Dependable Computer Systems (2005) COMMSTACK <FlexRay> 1.6 User's Manual,
Vienna, Austria.

Dependable Computer Systems (2006) COMMSTACK <FlexRay> 1.8 User's Manual,
Vienna, Austria.

Fennel, H., Bunzel, S., Heinecke, H., Bielefeld, J., Fürst, S., Schnelle, K.-P., Grote, W.,
Maldener, N., Weber, T., Wohlgemuth, F., Ruh, J., Lundh, L., Sandén, T., Heitkämper,
P., Rimkus, R., Leflour, J., Gilberg, A., Virnich, U., Voget, S., Nishikawa, K., Kajio, K.,
Lange, K., Scharnhorst, T. and Kundel, B. (2006) Achievements and Exploitation of the
AUTOSAR Development Partnership, Convergence 2006, October 2006, Detroit
Michigan, USA, SAE International, Warrendale, Pennsylvania, USA.

Fujitsu Microelectronics Europe (2007) Fujitsu FlexRay Driver Manual v1.3, Langen,
Germany.

Galla, T.M., Schreiner, D., Forster, W., Kutschera, C., Göschka, K.M. and Horauer, M.
(2007) Refactoring an Automotive Embedded Software Stack using the Component-
Based Paradigm, Proceedings of the International Symposium on Industrial Embedded
Systems, 2007. Lisbon, Portugal, July 4-6, IEEE Computer Society Washington, DC,
200-208.

Jackman, B. (2008) Class Notes on AUTOSAR, Waterford Institute of Technology.

TTTech Automotive GmbH (2007) AUTOSAR FlexRay Stack for Series Production,
Vienna, Austria.

Weka Fachmedien GmbH (2008) FlexRay and AUTOSAR get it right [online] available
at: <http://www.elektroniknet.de/home/automotive/autosar/english/flexray-and-autosar-get-it-right/> [accessed 15 May 2008].

LITERARY REVIEW

Zhu W. (2007) Performance Analysis of AUTOSAR Vehicle Network Gateways, unpublished thesis (M.Sc.), Waterford Institute of Technology.

Chapter 9 . Literary Review

Conclusion

9.1 Literary Review Summary

The chapters contained in the literary review should encompass all relevant information necessary to see the importance of this research. The reader should also be able to determine the strengths of the methodologies chosen. The review has not covered all possible methods available but has outlined popular approaches. This is due to the amount of possible areas to cover in these topics. To cover all possible areas would be unwieldy and add little to the review.

From the material covered it can be seen that the research previously conducted in the area of FlexRay has had a strong focus on the optimisation of the communication schedule. This can be seen from the research highlighted in Chapter 5. The hardware that is available also must pass conformance testing. The Bosch E-Ray is used in a number of devices and is supported by a number of different third party software products. The research that is outlined in this thesis focuses on the flow of data around a FlexRay node. This could help a developer to ensure a FlexRay node is optimised with respect to the timing of the system. This is an area which has little or no research conducted to date.

Different system analysis techniques were researched and a suitable option was chosen based on this. This is seen in chapter 5, and chapter 7 describes the chosen method in greater detail. From this a suitable software packages to carry out the intended research was also discussed and a decision made as the most suitable package commercially available. Finally all aspects of a FlexRay node were researched to ensure all relevant aspects of any node were included for the analysis that was conducted.

9.2 Available Literature

For the most part the information available for this research is good, accurate and accessible. There may however be a limit to the range of material available. For instance the FlexRay protocol is well defined and as such there are not many alternatives to the specifications laid out by the FlexRay consortium. The E-Ray chip is designed to be implemented in an FPGA and has been well documented by the designers. Manufacturer's datasheets on the specific implementations can therefore add little to the information outlined in the E-Ray user's manual. For simulation there are a number of well written books and other resources. These cover a multitude of different simulation methods and techniques. Performance analysis of software is also a big area of interest to companies. This is due to the ever present need to reduce cost. Also software development can be expensive and time consuming. Therefore there is a huge amount of material in the area of analysis.

For the most part the available literature is plentiful and well written. There are very few topics that can't be found in some document or book. There are also a number of papers or theses written on FlexRay scheduling and scheduling optimisation. These can be useful during the initial phase of learning.

9.3 Areas of Further Study

The necessary elements to complete this research have been covered in the literary review. It is necessary to be familiar with syntax of the programming language and methodology to write software programs. This is outside the scope of the thesis and therefore is left to the reader to gain knowledge and experience in this area if necessary. The automotive industry looks set to become more and more dependant on electric and electronic innovations. This means the area is rapidly changing and new products emerging. This leads to greater research possibilities and a wider area of study.

Section III:
Model Development

Chapter 10 . Methodology

10.1 Introduction

The unpredictability of circuits and systems increases with the size and complexity of the system. It is not possible to predict accurately how a system reacts to even a small change. The research presented here aims to develop a simulation model of a FlexRay node for analysis. The performance of the node, which includes buffer utilisation and throughput latency of data, will then be analysed. Recommendations for improvement can then be suggested based on observations made. This section will introduce the steps taken for successful simulation of a real world FlexRay node.

10.2 Simulation Process

In section 7.3 a simulation process was laid out. This process was followed for this research and the steps taken are outlined below (Banks et. al. 2001, p15-20). The process steps, as outlined by Banks et. al. (2001), are clear, concise and easily followed. They were also developed with a discrete event simulation model, (the modelling technique chosen for this research), and as such were followed as an appropriate methodology. Figure 10.1, is again the flow diagram for the simulation process (Banks et. al.2001, p16). Each step is discussed in turn.

MODEL DEVELOPMENT

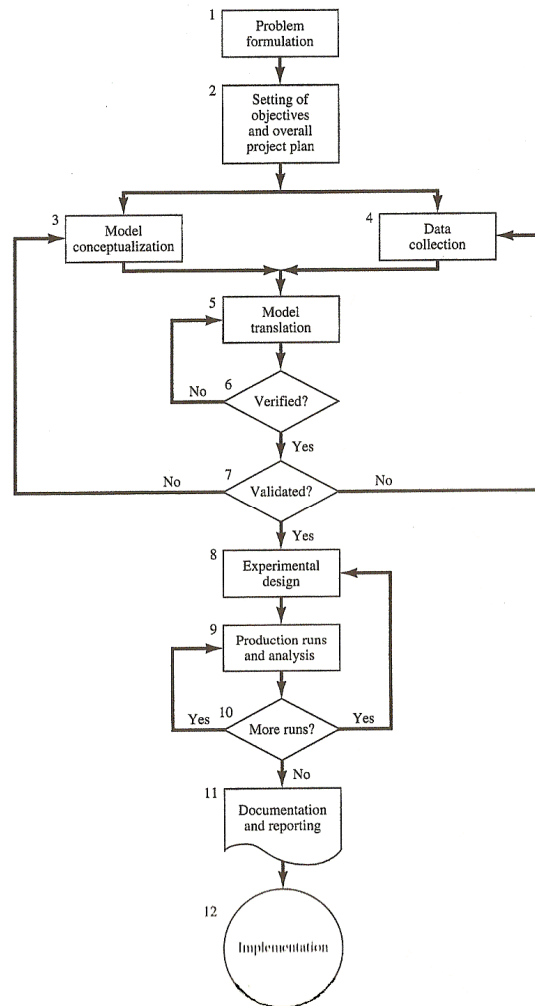


Figure 10.1: Simulation study steps

Problem Formulation: This is the first step of the simulation process. It is important that the analyst has a clear understanding of the problems that are to be addressed. This may involve discussions with any policy makers or stakeholders about the problems that may be faced for the system under investigation. The problem formulation may take time and problems may need to be re-examined many times before this step is complete.

Setting of Objectives and Overall Project Plan: At this stage it is decided if simulation is the correct procedure to address the problems. If simulation is found to be the correct action to take a project plan is devised. Alternatives to the different systems are devised and a method for analysing the suitability of these alternatives is defined.

MODEL DEVELOPMENT

Other considerations are constraints on time, people and the overall cost involved in undertaking the project.

Model Conceptualisation: During this step the fundamentals of the system under investigation are defined. The model should be kept as simple as possible with complexity added only as needed. This will reduce the costs of building the model. If possible the user of the model should be consulted during this process. This will create a better quality and more user-friendly model.

Data Collection: The earlier this stage is started the better the model will be. This is due to the time it takes to collect the data. The problem under investigation will determine the data to be collected. The data collected will include data from a real-world system. Data from the model may be collected at the calibration stage to analyse the accuracy of the model. The real-world data is used to calibrate and validate the model.

Model Translation: This step is where the model is converted from a conceptual object into a computer program. The type of simulation software is chosen based on a number of criteria. Some of the considerations when choosing the simulation software were discussed in chapter 7.

Verification: This stage of the simulation study involves debugging the model. The functionality of the model can be tested by passing in a set of inputs and checking the set of outputs obtained against a set of expected outputs. The model can be verified by testing various subsystems as they are constructed.

Validation: This step may need to be repeated many times. This step determines if an accurate representation of the system under investigation has been achieved. If the model cannot be deemed to be an accurate representation it may be necessary to go back to the data collection stage or model conceptualisation stage. Once the model accurately depicts the real world system over a number of scenarios it can be said to be validated.

Experimental Design: At this stage the experiments to be simulated are determined. This can be based on runs that have already be performed and analysed. The

MODEL DEVELOPMENT

considerations of this stage are, time taken to run or initialise the model and the number of repetitions of each run.

Production Runs and Analysis: The simulations measure of performance is determined during this stage.

The Need for More Runs: The focus of this stage is to determine if more runs of the model are necessary. Based on the analysis done at this stage the nature of any additional experiments are determined.

Documentation: The documentation of the simulation process is an important step. The documentation should be split between regular progress reports to any stakeholders and an overall model program report. The progress reports can be used to help any potential users to understand how the program works and clarify any misunderstandings. It also has the advantage of forcing the model builder to look at current progress and identify problems that arise. This can help the designer to meet deadlines. The overall program report will allow users to understand how the model operates and ultimately helps them to draw correct conclusions for the data obtained. It also allows modification of the model to suit other needs, if necessary, by other modellers.

Implementation: The success of this step is dependant on how well each of the previous steps were carried out. If the workings of the model are fully understood by the user and the model has been built with the problems under investigation in mind, the model should be a success. Likewise if the implementation has been impaired by a lack of understanding on the overall required outputs then the model may be deemed a failure.

10.3 Simulation Process in Relation to the Research

Problem Formulation: The problems faced by the adoption of FlexRay on a large scale were discussed in section 1.1 of this thesis. The problems covered were the main motivation behind the research outlined in this thesis.

Setting of Objectives and Overall Project Plan: The flow of data through a FlexRay node will depend on a number of factors such as the communication schedule or the buffering implementation. This would make running a wide range of scenarios on real world systems costly. The simulation of a node was chosen as a better alternative. A simulation model designed to analyse aspects of a FlexRay node will therefore be constructed and tested for suitability. The simulation model development flow process as defined in Banks et al. (2001, p16) will be used.

Model Conceptualisation: The features of the system relevant to the study were selected and defined. This includes the message RAM and handling of the communication schedule. Rejected aspects of the FlexRay node include the wakeup and startup phases. This is due to these phases having no impact on the flow of data through the node during communication.

Data Collection: The data to be collected was determined. All data from a real world system and the model were collected and analysed. The data that was collected included timing information for the software drivers to complete different tasks. Other data collected were timing constraints associated with the E-Ray communications controller implementation.

Model Translation: The model was built using SimEvents. This software was decided upon as it has all the required elements necessary to build a discrete-event model. It is also flexible as it allows the use of user defined elements or functions written in 'C' to be included. The steps involved in choosing the simulation program are outlined in section 7.10.

MODEL DEVELOPMENT

Verification: The model was verified in stages by testing the different subsections. When the model is completed it is verified as a whole. This stage was performed using SimEvents.

Validation: The model was validated against a small real world network. This included a number of different set of constraints to judge the performance of the model over a number of different scenarios. This was repeated until the model could be said to be validated. It was also evaluated to judge its ability to carryout its intended function.

Documentation: This was done at regular intervals. This is backed up by regular meetings with stakeholders in the research.

The following steps relate to the completed model as a tool. These are done to test real world scenarios where improvement in system performance is desired.

Experimental Design: The experiments are done as needed

Production Runs and Analysis: The experiments are run and the performance metrics obtained.

The Need for More Runs: This is done as needed.

Implementation: The real world system can be modified and improved based on the output of the simulation experiments.

10.4 Conclusion

The FlexRay protocol is new and emerging. This leads to a wide range of research opportunities to improve FlexRay products and systems. The complexity of setting up a FlexRay node makes it difficult to analyse over a wide range of constraints. The cost to set up an adequate real world FlexRay network to test a hypothesis can also become prohibitive. As it is not always possible to observe real world systems a simulation of a FlexRay system is a more effective and viable solution. Using the methodology outlined in this chapter an effective model should be achieved.

10.6 References

Banks, J., Carson, J. S., Nelson, B. L. and Nicol, D. M. (2001) Discrete-Event System Simulation, New Jersey: Prentice Hall.

Chapter 11 . Simulation Model Development

11.1 Introduction

The simulation model was designed to accurately represent a real world FlexRay node. To achieve this, the model was based on the Bosch E-Ray communications controller. This chapter will outline the specification of the FlexRay node that defined the structure and performance of the simulation. The model will be broken down into its various subsystems. The functionality of these subsystems will be explained.

This model used MATLAB, Simulink and SimEvents to build, verify and validate the model. The various operation and performance characteristics of the components used will be described as necessary throughout this chapter. Due to the size of the model developed, only a small number of the main subsystems will be described. Figure 11.1 (Banks et. al. 2001, p16) highlights the stage at which the model development occurs in the model development cycle. At this stage research into the operation of the system has been done and the simulation model is built.

MODEL DEVELOPMENT

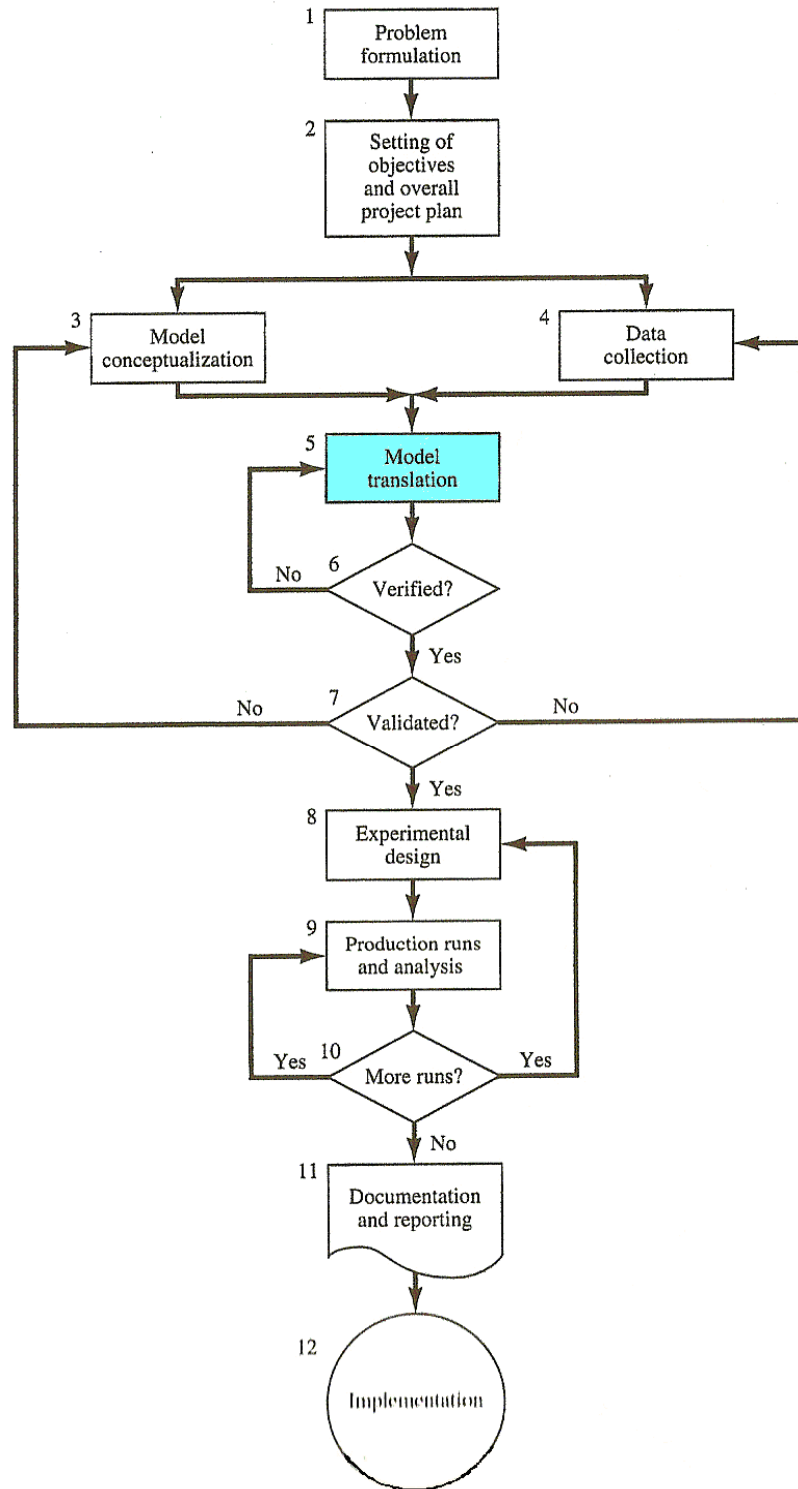


Figure 11.1: FlexRay development steps

11.2 Specification Development Process

When the specification for the model was developed, it was based on the specifications of the separate components making up a FlexRay system. The structure of a node based system can be seen in Figure 11.2.

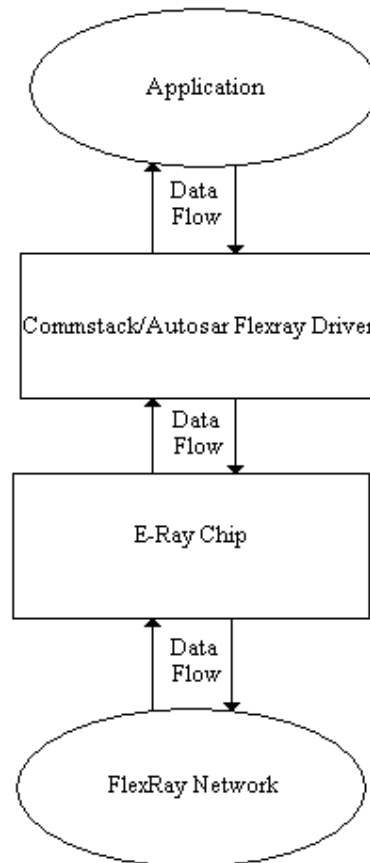


Figure 11.2: FlexRay node elements

The elements of the node are the application, the software driver and E-Ray communications controller. The FlexRay network represents the physical bus over which data is transmitted. It was necessary to understand how each of these layers work and interact to accurately reflect the workings of a FlexRay node.

By viewing a FlexRay based system in the divisions shown in Figure 11.2, the build process was more easily modularised. This meant that each element of the system could be built and tested separately. The final model then consists of these individual subsystems. This method allows each section to be built to the specifications applicable to it. It does mean however that there may be some work needed when connecting the elements of the overall system together. Problems can arise in the form of syntax issues,

i.e. attribute names vary slightly, or in the form of a system receiving an entity it cannot handle.

11.3 Simulation Model Specifications

Each of the nodes' elements was considered in relation to their inputs/outputs and functionality. Figure 11.3 shows the inputs, outputs and considerations that were necessary to be investigated for the application layer.

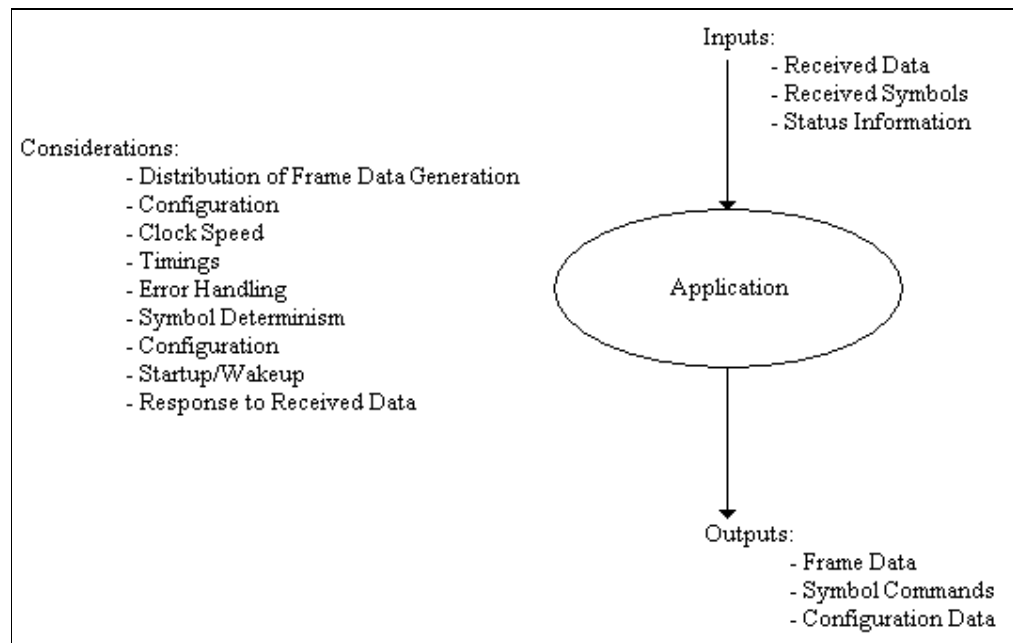


Figure 11.3: Application inputs, outputs and considerations

As can be seen the considerations that were examined were elements that affect the speed and flow of data. For each section a number of considerations were investigated. The remaining sections of the FlexRay node considerations can be seen in Figures 11.4, 11.5 and 11.6.

MODEL DEVELOPMENT

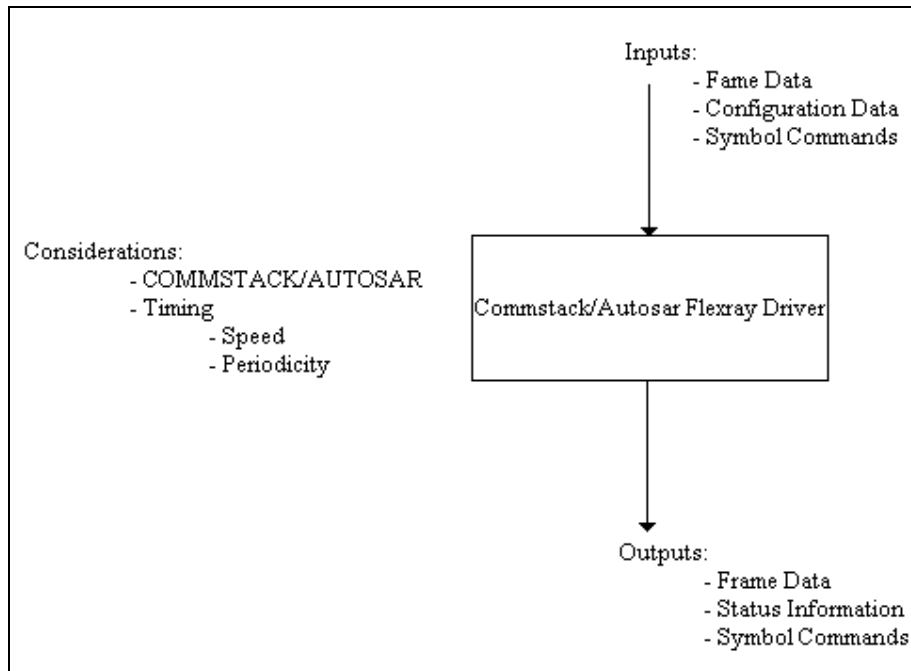


Figure 11.4: Software driver inputs, outputs and considerations

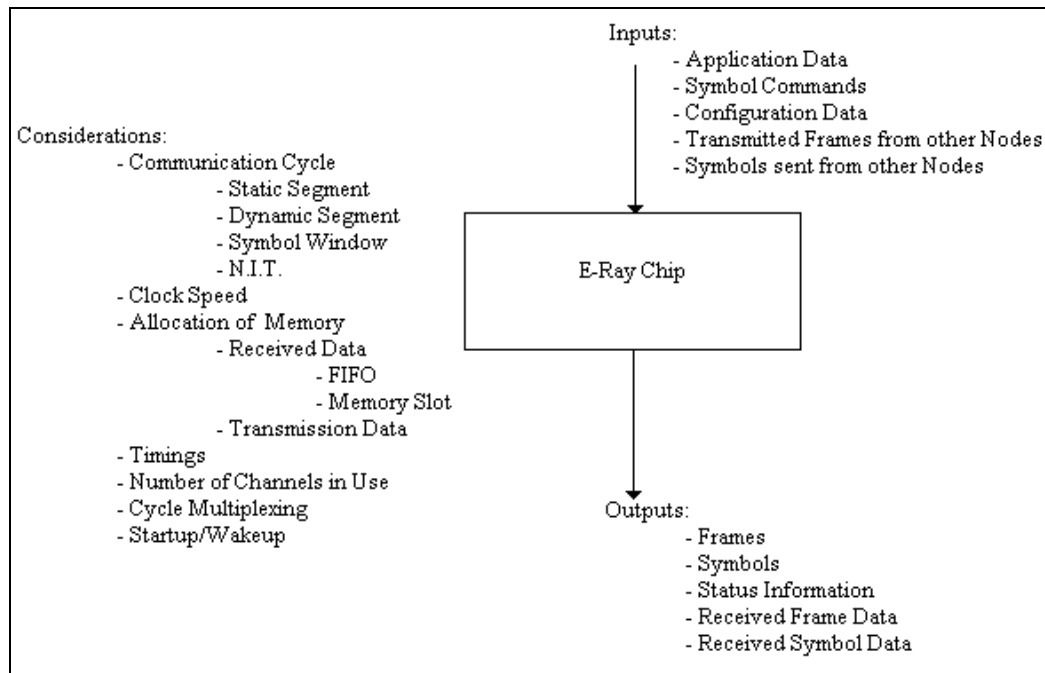


Figure 11.5: Communications controller inputs, outputs and considerations

MODEL DEVELOPMENT

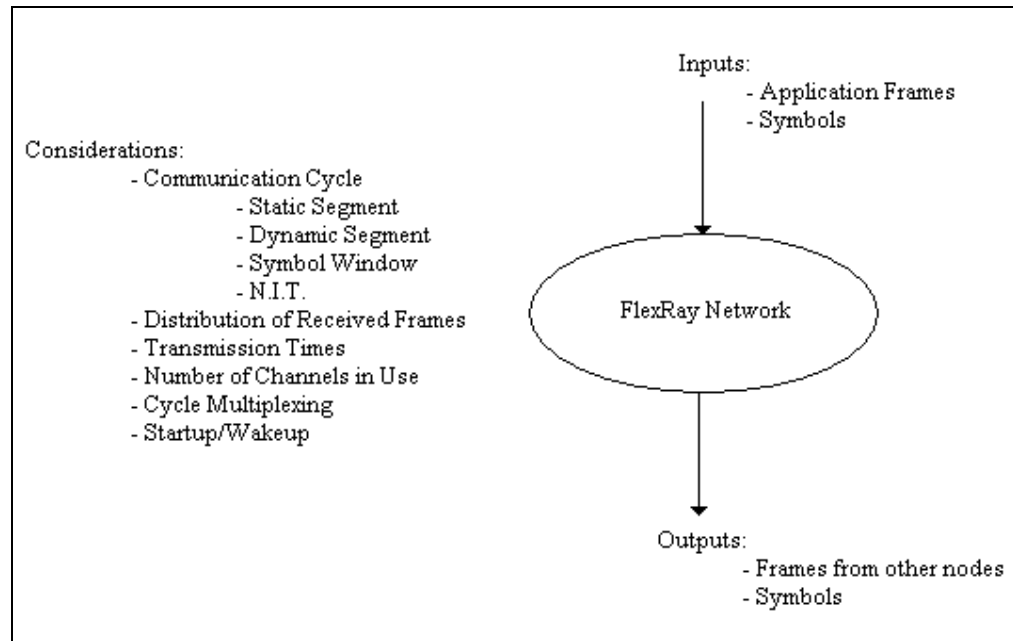


Figure 11.6: Physical bus inputs, outputs and considerations

After this process it was decided that the startup and wakeup processes would have no effect on the flow of data through a communicating node. Therefore it was felt that these two processes would have no bearing on the outcome of the research. All the other considerations as outlined in Figures 11.3 to 11.6 could have a big impact on the flow and timing of data. For this reason it was necessary to accurately depict them in the model.

The next step was to investigate the workings and makeup of those elements outlined above and to implement them in the simulation model.

11.3.1 The Model Design Philosophy

Each of the four basic elements that make up a FlexRay node (application, software driver, communications controller as well as the physical bus layers) may be made up of a number of subsystems. The simulation model was therefore designed to accurately reflect these divisions. This section will outline how each of the sections is divided up and their subsequent implementation in SimEvents.

The two diagrams Figures 11.7 and 11.8 show the top layer of the simulation model and the main elements of the FlexRay node model. The design of the top layer is used to give a more realistic breakdown of the model. However it provides no functionality to the model.

MODEL DEVELOPMENT

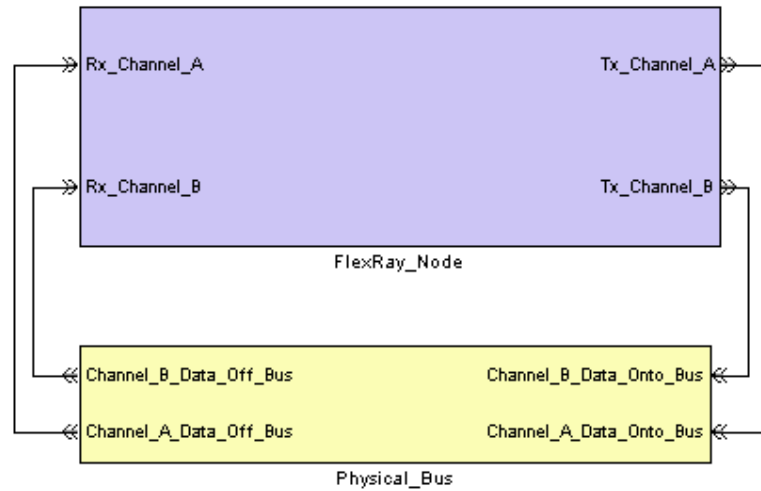


Figure 11.7: Top layer of simulation model

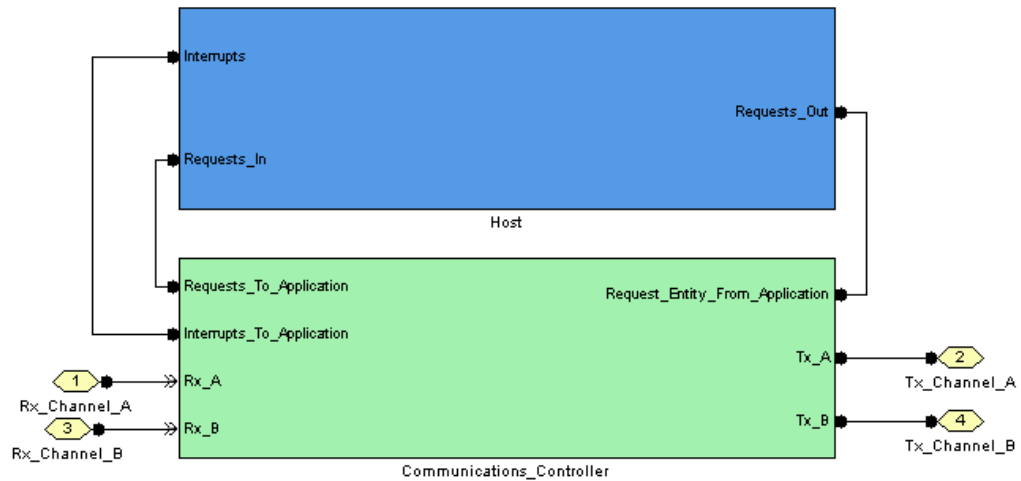


Figure 11.8: FlexRay model subsections

When designing the blocks and subsystems of the model the guidelines as defined by MAAB (Mathworks Automotive Advisory Board [MAAB] 2007) were used. These guidelines are for use when modeling with MATLAB, Simulink and Stateflow. Therefore not all the guidelines were useable without minor alterations. The basic workings of each section of the model are discussed briefly later in section 11.5. For a more complete explanation of the workings of FlexRay, E-Ray and the FlexRay software drivers see chapters 4, 6 and 8 of this thesis.

MODEL DEVELOPMENT

To make an adaptable model a number of entities types were designed. Each entity had a number attributes associated with them. These entities were used to represent the main signals of concern to the study. The entities that were developed were:

- Requests: These entities represent the signals and data sent between the communications controller and the host of the node. The host is a combination of the application layer and the software driver.
- Slots: These are in the form of static, dynamic and minislots. They define when slots begin.
- Cycle: This is used to represent when a new communication cycle begins. This is used by various other subsystems to identify the current time also.
- Frames: These entities represent the data to be transmitted over the physical bus.

The various paths that the different entities could take though the model are shown in Figure 11.9. In Figure 11.9 R= request entity, S= slot entity, F= frame entity and C = communication cycle entity. The E-Ray module abbreviations shown are the Controller Host Interface (CHI), Input Buffer (IBF), Output Buffer (OBF), Protocol Register (PRT) and Transient buffer (TBF). Message is abbreviated as MSG.

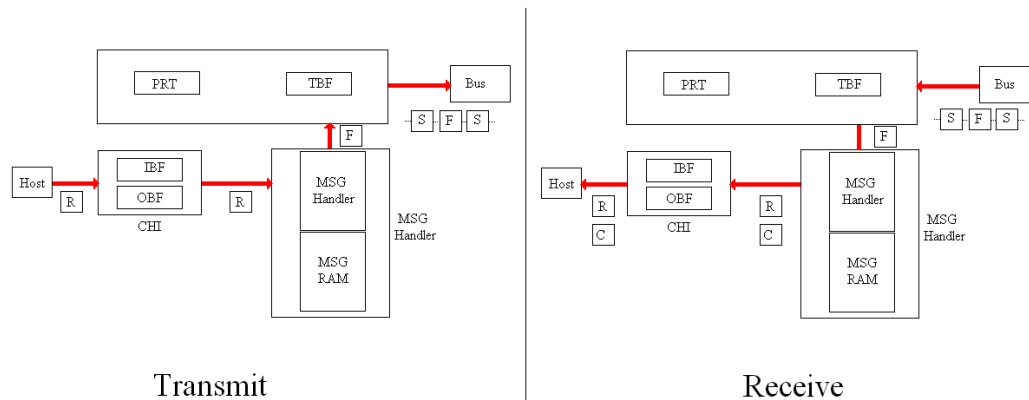


Figure 11.9: Entity Paths

As can be seen there are a limited number of paths that any one type of entity may take. This will be discussed in section 11.3.2

It was also decided that colour coding the model would make debugging the model easier and quicker. To do this a colour coding convention was developed. This colour coding chart can be seen in Appendix A.

11.3.2 Entity Types

As was stated in section 11.3.1 there are four basic entity types that were developed to model a FlexRay system. These represent the signals that are of interest to this research. These signals are in the form of timing (slot and communication cycle entities) as well as data signals (request and frame entities). Figure 11.9 also showed that there are two basic directions that these entities can take i.e. transmit and receive. Figures 11.10 -11.13 show a flow of the slot, request, frame and cycle entities throughout the model.

Figure 11.10 shows that the generation of slots is based on synchronisation entities that are produced by the synchronisation block. The generation of slots is done in the global time unit subsection of the model. The current slot entity is then passed to a number of blocks. This allows the message RAM to check for any frame that would be sent during the next slot. The media access control block also uses a slot entity to ensure that the current frame in the transfer buffer is the next valid frame before allowing the controller to commence communication. Finally the physical bus uses the current slot entity to check if slot is assigned to the simulated node. If the slot is assigned as a receive slot for the node then there is no possibility that the physical bus will generate a frame during that slot. If the slot is not assigned to that node, the physical bus has a possibility to generate a frame.

Any slot that is received from the physical bus is used by the global time unit to create a dynamic slot if the static segment is complete. The generation of the dynamic slot occurs at the generation time of a minislot if a slot has been received from the physical bus. This means that while transmission is occurring, during the dynamic segment, the current slot entity must be blocked from entering the communications controller model.

MODEL DEVELOPMENT

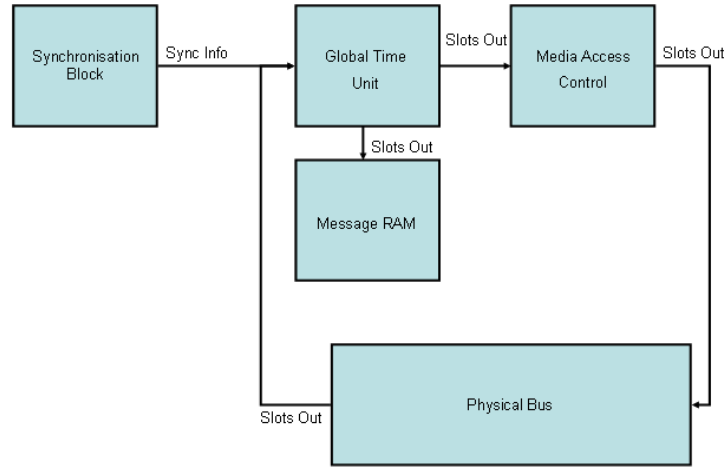


Figure 11.10: Slot entity paths

In Figure 11.11 the request entity paths can be seen. These entities are generated from the application layer. These requests could be in the form of asking for a stored frame in the message RAM of the communications controller. It could also be a request to update a buffer assigned to a transmission frame. These requests must pass from the application layer into the driver. The requests will then pass through the appropriate layers of the communications controller.

As can be seen from Figure 11.11 the requests can travel down from the application to the communications controller. Requested data will then travel back through the communications controller and software driver if necessary.

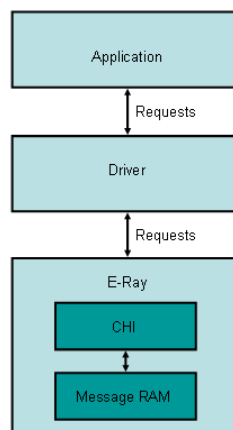


Figure 11.11: Request entity paths

Frame entities represent any frame that is to be transmitted on or successfully received from the physical bus layer. They pass through a number of different blocks.

MODEL DEVELOPMENT

This is shown in Figure 11.12. They are stored in the message RAM and passed onto the bus at the appropriate time using the media access control subsystem. If a frame is generated by the physical bus layer and it matches the filtering criteria, the frame is stored in the message RAM. Another entity could have been created to represent the buffer entities. These would contain the same information as the frame entities. This means that there is another type of entity used in the model but was never formally defined.

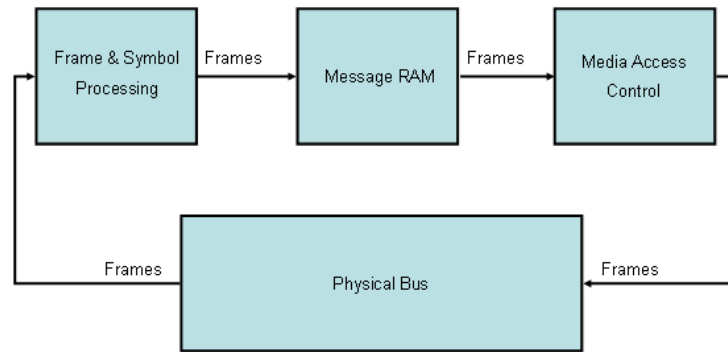


Figure 11.12: Frame entity paths

The final entity type is the cycle entity. This defines the start of a new communication cycle. It is passed up the application layer as an 'interrupt' to give the application layer a time reference. The global time unit subsystem receives this entity at the start of the symbol window/network idle time. This is then used to indicate that dynamic slot generation should cease. The paths the cycle entity will take are shown in Figure 11.13.

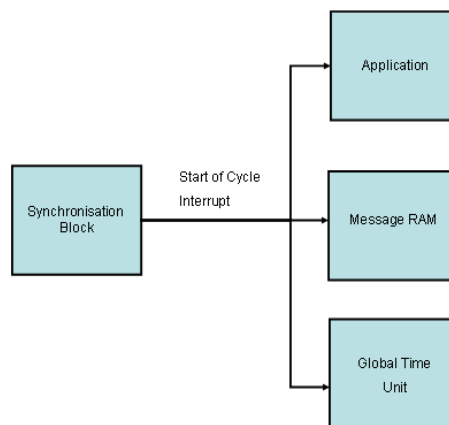


Figure 11.13: Cycle entity paths

MODEL DEVELOPMENT

Table 11.1 describes all the entity attributes. It includes the static slot and minislot entities that are used to define the start of slot entities. The static slot, minislot and cycle entities encompass all of the entities necessary to perform the synchronisation of the model. This is further described in section 11.5.1.1.

Type	Attribute	Attribute Number	Notes
Slot			
	Slot_No	1	Used to identify a particular slot
	Remaining_Mini	2	Used to calculate the remaining transmission time in the physical bus.
	Cycle_Count	3	This represents the current communications cycle
Requests			
	Request_Type	1	1 = get frame for transmission. 2 = get frame for the host. 3 = update frame attributes. 4 = read from the FIFO
	Frame_ID	2	The frame to be updated, transmitted or sent to the host.
	Data	3	The update data if any.
	Channel_Config	4	The channel the message is to be transmitted on if necessary.
	TX_Type	5	1 = The frame should be transmitted. This is used when updating a message buffer for transmission. 2 = the frame was received and should not be transmitted
Frames			
	Frame_ID	1	Used to identify a particular frame
	Cycle_Code	2	Used for filtering if desired.
	Channel_Config	3	1 = transmit/receive on A. 2 = transmit/receive on B. 3 = transmit/receive on A & B. This is the same for Request Entities.
	Data	4	The data length of the frame
	TX_Type	5	1 = The frame should be transmitted..
Static Slots			Describes the actual static slot
	Slot_Number	1	Used to identify a particular slot
	Remaining_Mini	2	This is set to ensure that the physical bus isn't blocked for generating frames during the static segment
Mini Slots			Describes the current minislot
	Mini_Slot_Number	1	Used to identify the number of minislots that have expired during the current dynamic segment
Cycle Count			
	Cycle_Count_Number	1	Holds the current cycle count, values range from 0-63

Table 11.1: Entity attributes

11.4 Model Metrics

Before metrics can be developed from the output of the model an analyst must be able to obtain measurements from output. The following sections will describe the measurements that can be obtained from the model.

11.4.1 Data Flow Measurements

It has been stated that the model views the FlexRay system as a flow of data through the system. An example of this is illustrated in Figure 11.14. In Figure 11.14 the small squares within the separate model components represent the various delays associated with the system. The messages must pass through each of these to reach their destination. As can be seen there are two paths that the data may take. The data may originate on the communications bus. This represents data sent by the various nodes of the system. It will take time to propagate over the bus and into a node. The data will then work its way up through the communications controller and the software driver to the application layer. The data can then be processed.

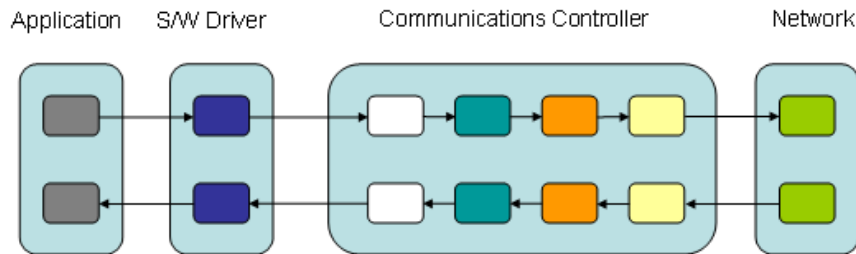


Figure 11.14: Model as a flow of data

Figure 11.14 shows how there can be one or more delays associated with any one layer of the model. The implementation of these layers will have an effect on the performance and operation of the system. For instance the manner in which the application handles data from the communications controller will effect how quickly incoming data can be processed. The implementation may be seen only as one delay or as many delays and this will depend on the setup and characteristics of the layer in question. By analysing the paths better conclusions can be made.

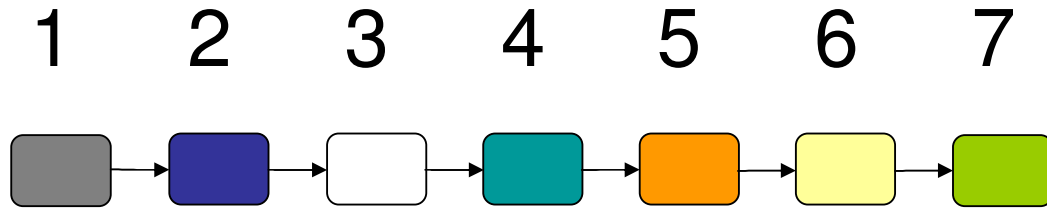


Figure 11.15: E-Ray data flow path

Figure 11.15 shows one path that data may take through the E-Ray chip. The diagram is numbered as follows:

1. Data moves off the bus and is placed in the protocol controllers.
2. The message then moves into the transient buffer.
3. From here the message is passed to the message handler to be placed in a Message Buffer.
4. The message remains in the Message Buffer until requested by the host.
5. The message is passed into the output buffer by the message handler.
6. The message is passed from the output buffer to the software driver.
7. Finally the message passes to the application layer for processing.

The path will experience extra delays as the data moving in one direction could affect the movement of the data in the opposite direction. The messages stored in the message RAM will also spend a potentially indeterminate amount of time in a buffer before being read or transmitted. By recording data entering various stages and the time when it is passed onto the next stage a clear view of the data flow can be achieved. This will help to identify why, if any, deadlines are missed, or why a message is overwritten before it is processed or transmitted. Any ‘bottlenecks’ in the system can then be analysed and a solution proposed.

In Figure 11.2 the overall conceptual view of a FlexRay node was shown. The data flow through such a system is also shown in Figure 11.14. Various aspects of a FlexRay system could be of interest to any particular systems analysis. For the model to be a useful as a performance analysis tool it must return suitable measurement values. Once these metrics were determined the model could then be modified to return these values.

The first consideration taken into account was the time base of FlexRay. This was taken as a real world value directly from the simulation clock. This would be easy to achieve and all measurements could then be easily time-stamped and saved as MATLAB workspace variables. The time stamps could then be compared to the

expected start times of slots or communications cycle start events. This would give an accurate view of when any simulation event occurred.

Another consideration taken into account was the type of analysis that was required for the model. The function of the model would have a big effect on the type of metrics that should be obtainable from the model. For instance the focus of this model was on the flow of data through a FlexRay node. This means that the power consumption of the node could not be measured and any metrics based on this could be ignored in this study.

The final consideration was to attempt to record as many possible measurements that may be required by a user of the model. This would then make the model as useable for as many tests and testing situations as possible.

When these were all taken into account the metrics could be developed and analysed for suitability. The model could then be constructed to accommodate any measurements necessary to achieve the required flexibility. For more information on performance analysis and metrics see chapter 5 of this thesis.

11.4.2 Application Measurements

Figure 11.3 shows the considerations for the application layer of the model. The application layer takes in data from and passes data to the software driver. This data can be seen as data from the communications controller. The role and operation of the driver will be introduced in section 11.4.3. The information passed to the driver is either configuration data during startup or data for transmission after the startup phase to/from the communications controller. As the model is not concerned with the startup phase data to and from the model application layer is only concerned with communication data. The main concern with the application would therefore be the number of frames[§] it can process within a given communication cycle.

The number of frames that the application processes within a given communication cycle could give an insight into the suitability of the setup. For instance

[§] A frame is not completely passed on by the host. For any given static slot the host merely updates the data section in memory RAM allocated to a particular frame and the header section remains unchanged. For more information see chapters 4 and 6 of this thesis.

a node could be set up to read a sensor and output data as well as calculate a value based on received data and output this calculated value. If the application is not able to transmit the data to the communications controller within a given time the data may not be transmitted during the current communication cycle. This could indicate that the application coding takes too long to execute or that the communication schedule is not appropriate. Likewise if the application executes very rapidly the node's host controller may be idle for long periods. This could mean that an extra application may be added to that node (hardware limitations such as memory requirements permitting).

Due to these considerations it was decided that to measure:

- The number of frames sent to the communications driver.
- The number of incoming frames serviced.
- The overall execution time of the application.

11.4.3 Software Driver Measurements

Like the application, the software driver is concerned with data for transmission and data received to/from the application and communications controller. This can be seen in Figure 11.4. However unlike the application the driver cannot generate data. It merely accepts data and passes it between the communications controller and the application. In this way the most significant effect that the driver will have will be to slow down the data as it waits for the communications controller or application to respond to commands.

The measurements that can be drawn from this layer are:

- The wait time of a frame to be passed from the host to the communications controller.
- The wait time of a frame to be passed from the communications controller to the host.

11.4.4 Communication Controller Measurements

The communications controller considerations are shown in Figure 11.5. This is an important aspect of the model. This layer is where the research conducted in the thesis is focused. The communications controller is responsible for ensuring that frames are only transmitted during their allocated slots. It must also accept frames sent over the physical bus if those frames match the acceptance filter criteria. Both transmit and

MODEL DEVELOPMENT

receive frames must be stored in the communications controller until ready to be transmitted or requested by the application.

To be able to measure the performance of this layer it is necessary to know:

- What data is coming into and out of the controller.
- The channel(s) a frame is transmitted/received on.
- The frames that are passed and rejected by the filtering for the FIFO and normal filtering.
- The number of frames stored in the FIFO.
- The average wait time for a message stored in the message RAM should also be recorded.
- The number of requests from the application layer that are serviced by the controller.
- The values for the various slots and communication segments.

11.4.5 Physical Bus Measurements

The physical bus transmits frames received from any communications controller. The communications controller must ensure that it only transmits data during its allocated slots. In this way the model of the physical bus must be able to generate frames to simulate traffic from other nodes. The physical bus must not block the frames from the communications controller model unless it is the dynamic segment. In the dynamic segment frames can occupy as many minislots as necessary. Figure 11.6 again shows some considerations that should be taken into account.

The physical bus is completely dependant on the communication cycle. As such all the measurements should be considered in terms of the communication cycle. If possible however the different segment times should be taken into consideration. In this way the bus loading for the static and dynamic segment can be viewed separately.

The measurements therefore that could be taken are:

- The frames received from the communication controller.
- The frames generated and transmitted as other nodes on the bus.
- The average wait time that the frame occupied the bus.
- Each measurement should be split up to represent each of the two communication channels.

11.4.6 Model Metrics Summary

To make a useful and adaptable model a number of items were investigated. In this section the metrics that may be desired were discussed. There are a number of possible metrics that an analyst may want. The model was therefore designed to provide as many of these as possible. A number of measurement blocks were added to the model. This will be seen in section 11.5. A list of all the recordable values and variables for the model can be seen in Appendix B of this thesis.

11.5 The Model

11.5.1 Communications Controller Model

The communications controller will be modelled on a Bosch E-Ray communications controller chip. This meets the specifications laid out in the FlexRay specifications. The divisions and functional blocks of a communications controller can be seen in Figure 11.16 (Robert Bosch GmbH 2006a, p14). Figure 11.16 is the block diagram of an E-Ray chip. The simulation model of the communication controller was designed to reflect the layout of these diagrams.

The various sections of Figure 11.16 are as follows:

- IF: interface.
- IBF: the input buffers.
- OBF: the output buffers.
- TBF: transient buffer RAM.
- PRT: FlexRay channel protocol controller.
- GTU: global time unit.
- SUC: system universal control.
- FSP: frame and symbol management.
- NM: the network management.
- INT: interrupt handler.

MODEL DEVELOPMENT

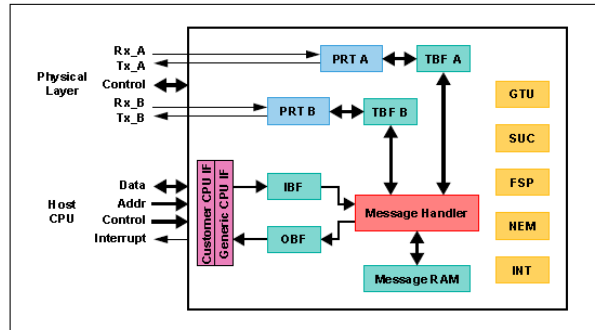


Figure 11.16: E-Ray block diagram

Figure 11.17 shows the functionality that was required for the communications controller layer of the model. This was based on the specifications of the E-Ray controller and the FlexRay specifications. The outer circle show the areas of interest to this research.

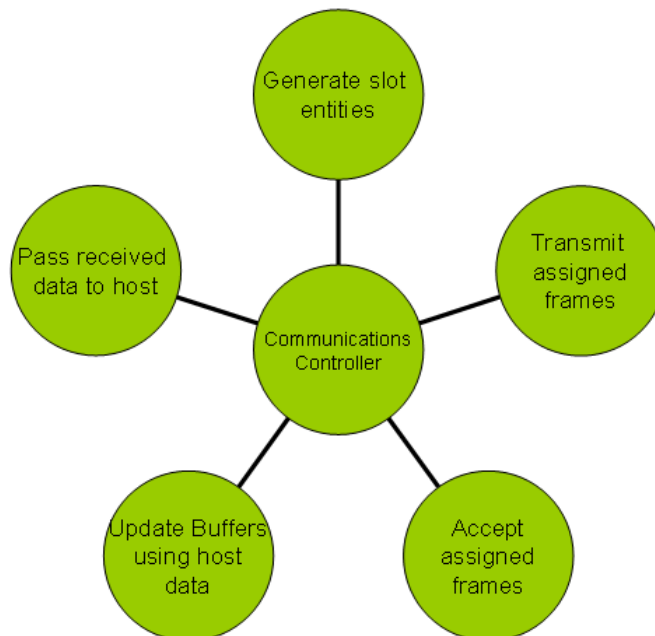


Figure 11.17: Communications controller tasks

The model representation of the E-Ray chip can be seen in Figure 11.18. The naming of the sections is based on that of Figure 11.16. This was to achieve a higher level abstraction more consistent with that of the FlexRay protocol. For instance the input buffers and output buffers form part of the controller host interface. By naming the model subsystem input buffer and output buffer it may not be clear to all users of the model immediately that the buffers were used for transfer between the host and the

MODEL DEVELOPMENT

communications controller and not onto the physical bus from the communications controller. Instead the use of ‘Media_Access_Control’ and ‘Frame_And_Symbol_Processing’ was used.

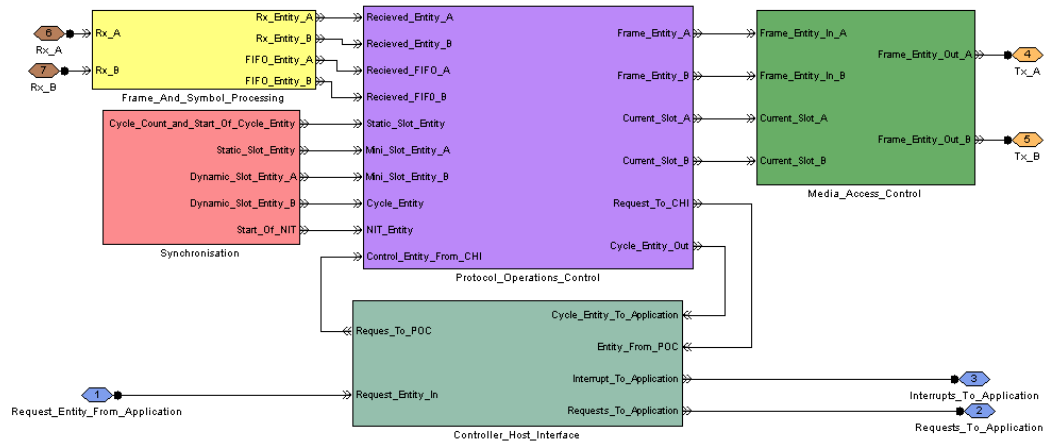


Figure 11.18: Model of the communications controller

This naming convention was used for the following reasons. The paths that a message may take through a E-Ray communications controller is show in Figure 11.16. The subsystems coloured yellow, in Figure 11.16 help to maintain the accurate running of the node. They have little bearing on the flow data through the node with the exception of the frame and symbol processing and the global time unit subsystem. The frame and symbol processing subsystem accepts or rejects the frames arriving and allows the node to transmit the frame only during its current slot. The global time unit merely tells the communications controller the current global time.

The role of the transient buffer RAMs (there is one for each channel) is to store any messages to be sent out on the bus as well as the most recent message received on the bus. Figure 11.19 shows the structure of the single transient buffer RAMs (Robert Bosch GmbH 2006, p138).

MODEL DEVELOPMENT

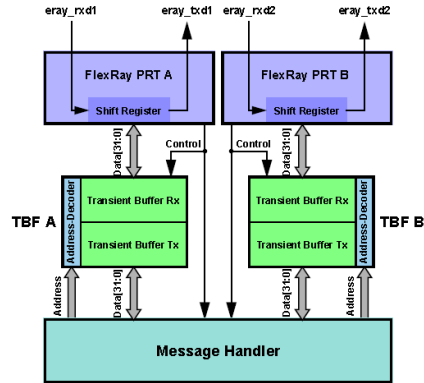


Figure 11.19 Transient buffer RAM structure

As there are separate dedicated receive and transmit buffers it is possible to split these components up into two separate functional blocks. Therefore the transmit buffers become the media access control and only allow the correct message to be sent out. The receive buffers then become the frame and symbol processing subsystem as they only send the data to the message handler if the received frame matches filter criteria.

11.5.1.1 Synchronisation Model

The synchronisation block, Figure 11.20, acts as part of the global time unit of the E-Ray chip. It carries out the same function as the macrotick generation and clock synchronisation processing blocks as defined by the FlexRay specification (2005). It forms along with the 'Global_Time_Unit' subsystem in the 'Protocol_Operations_Control' subsystem the whole clock synchronisation and time handler. The function of the synchronisation subsystem is to give a reference for when slots begin.

MODEL DEVELOPMENT

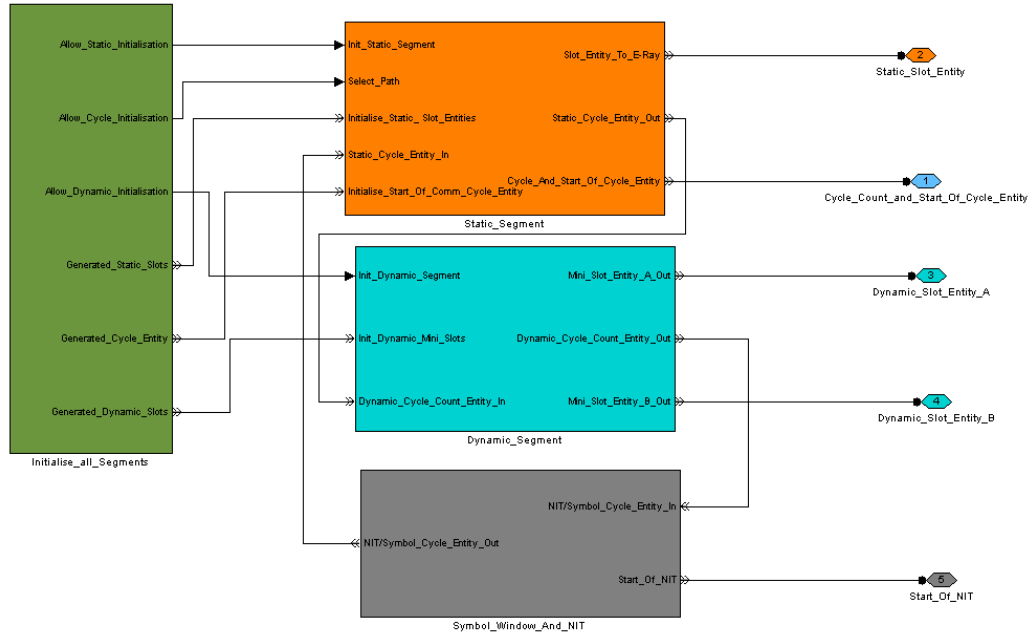


Figure 11.20: Synchronisation block

The synchronisation subsystem as can be seen in Figure 11.20, is divided into four main sections. These sections are the ‘Initialise_All_Segments’, ‘Static_Segment’, ‘Dynamic_Segment’ and the ‘Symbol_Window_And_NIT’ blocks. The function and operation of these blocks will be discussed below.

The synchronisation subsystem is controlled by a cycle entity that is passed between each lower level subsystem in turn. When a subsystem has the cycle entity it is allowed to perform its function. For example, if the static segment subsystem is in possession of the cycle entity it is allowed to generate static slots. After a predefined time the cycle entity is passed to the next subsystem and this subsystem then performs its function. The flow of the cycle entity can be seen below in Figure 11.21. Also at the start of each cycle the cycle entity is passed up to higher layers of the model.

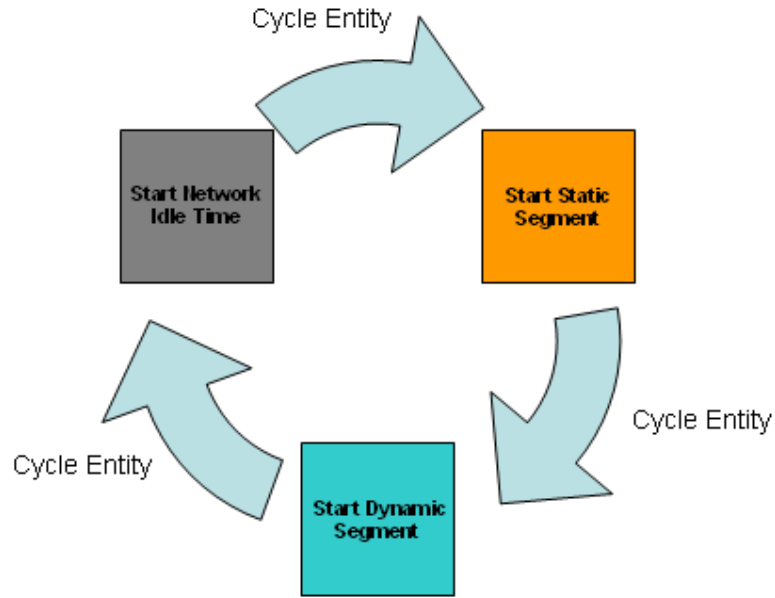


Figure 11.21: Cycle entity flow diagram

11.5.1.1.1 Synchronisation Initialisation Block

This block sets up the static and dynamic segment subsystems. It also generates the cycle entity that is used to control the operation of the different subsystems. Each initialisation segment is confined to a different subsection. This is again to aid readability and to avoid confusion. Figure 11.22 shows the subsystems used in the initialisation subsystems.

MODEL DEVELOPMENT

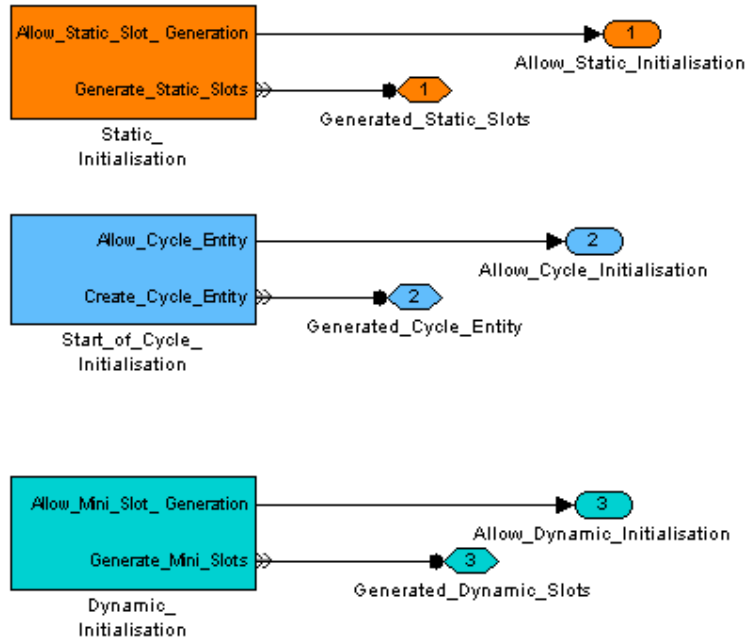


Figure 11.22: Initialisation block

The static initialisation generates static slot entities that will be used to determine the current static slot in the static segment subsystem. This is shown in Figure 11.23. Each entity is generated using the SimEvents time-based generation block. This means that at fixed, evenly spaced points in time a new entity is created.

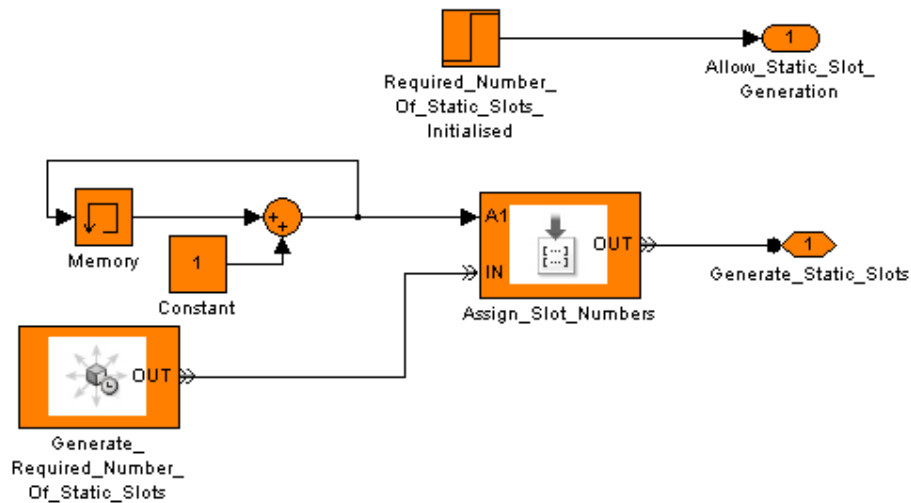


Figure 11.23: Initialise static segment block

MODEL DEVELOPMENT

Each generated static slot has an attribute assigned to it. This attribute is the 'Slot_Number' attribute. Its value is obtained from the memory block that adds one every time an entity passes through the 'Assign_Slot_Numbers' block. Once the number of static slots has been obtained the 'Required_Number_Of_Static_Slots_Initialised' block is asserted. This will cause the generation process to be blocked. This model block is based on time but is set using workspace variables.

The dynamic initialisation subsystem generates minislots that will be used to determine the current dynamic slot during the dynamic segment. This subsystem operates in the same way as the static slot initialisation subsystem. When the required numbers of minislots are obtained a signal is asserted and this causes this subsystem to be blocked from generating more entities.

The subsystem that initialises the cycle entity works in a similar way to the static and dynamic initialisation subsystems. This subsystem differs from the static and dynamic initialisation subsystems as it only generates one entity. As such the set attribute block does not need an external signal to indicate what the attribute value should be. This is instead 'hard-coded' using a dialog box in the parameters of the set attribute block.

11.5.1.1.2 Static Segment Subsystem

In this subsystem, shown in Figure 11.24, the static slots from the initialisation subsystem are stored in a FIFO. When the cycle entity arrives it enables slots to move from the FIFO into a server. Before they are moved into the server however they are replicated and sent to higher levels of the model. The server holds the static slots each for the desired length of a static slot. The cycle entity should therefore only allow enough time for the required number of static slots to be replicated and sent out.

After the slot entity has left the server it is sent back to the FIFO so that during the next cycle the process can begin again. This eliminates the need to continually generating slot entities.

MODEL DEVELOPMENT

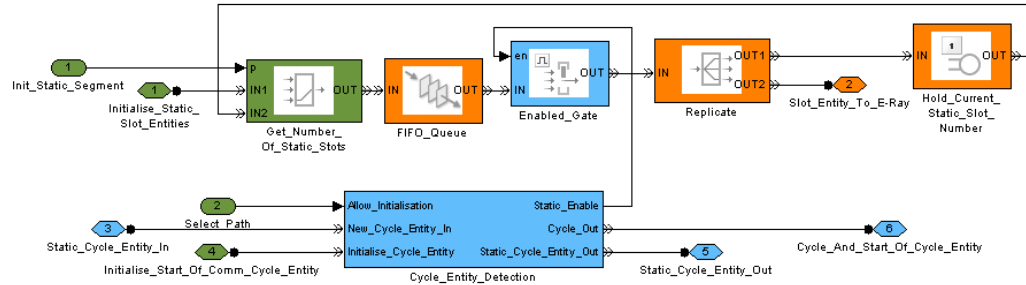


Figure 11.24: Static segment block

Figure 11.25 shows the cycle entity detection subsystem elements. This subsystem accepts the initial cycle entity. It then replicates the entity and sends it to higher subsystems that it may require the entity to function correctly. The cycle entity will then enter the server. This server holds the cycle entity and creates a signal that allows the enable gate in the static segment block to pass slot entities.

It is necessary to have a second server to avoid timing issues. After the last slot has passed from the FIFO the cycle entity passed to the second server. This server holds the entity for the remaining time of the static segment. It was found that an extra static slot entity could pass from the FIFO arbitrarily if this second server was not employed. If an extra slot entity passes from the FIFO then errors occur based on the slot numbers.

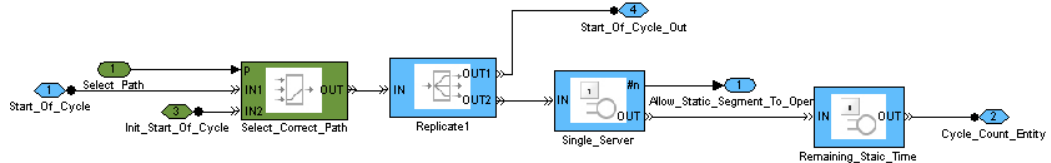


Figure 11.25: Get start of cycle

11.5.1.1.3 Dynamic Segment Subsystem

The dynamic segment operates in a similar way to the static segment. Entities are allowed to be generated when an enable signal is asserted. This signal is asserted when the cycle entity enters the 'Cycle_Entity_Detection' subsystem. It can be seen in Figure 11.26 that there is a replicate block within this subsystem. This block takes the generated minislot and passes it to two 'out ports'. This allows for the system to keep

MODEL DEVELOPMENT

track of the current dynamic slot on each channel separately. This necessary due to the workings of the ‘Global_Time_Unit’ subsystem and will be discussed later.

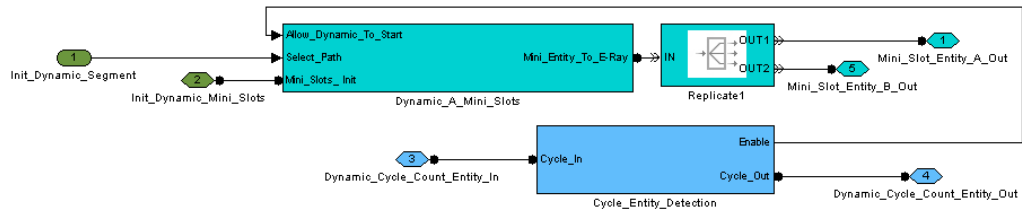


Figure 11.26: Dynamic segment block

The minislot generation takes place as shown below in Figure 11.27. As can be seen it operates in the same way as the static slot generation.

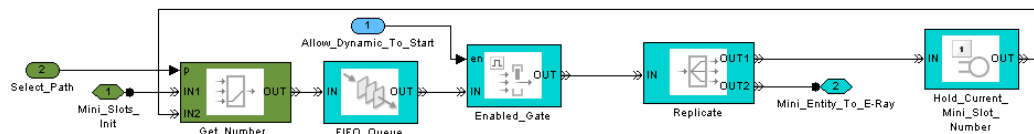


Figure 11.27: Dynamic channel block

The enable block for the dynamic segment is less complicated than for the static segment. This is because it has no routing blocks for the initialisation stage or a replication block section for the cycle entity. It does however utilise the two server approach to eliminate any erroneous minislot generation errors. Therefore the only required elements are those only shown in Figure 11.28.

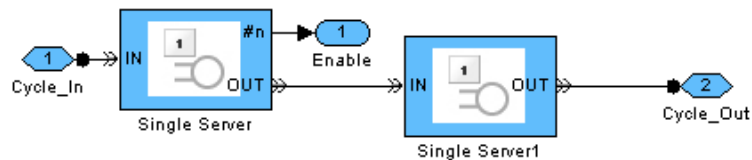


Figure 11.28: Dynamic enable block

11.5.1.1.4 Symbol Window and NIT Subsystem

The symbol window and network idle time (NIT) subsystem holds the cycle entity for the remainder of the communication cycle. It also increments the cycle count attribute. Figure 11.29 shows the blocks used to achieve this. The cycle count attribute is obtained from the entity when it enters the block. When it leaves the block an updated cycle count replaces the old value. On entering this subsystem, the cycle count entity is replicated and sent to the global time unit subsystem. This will then stop dynamic slot generation.

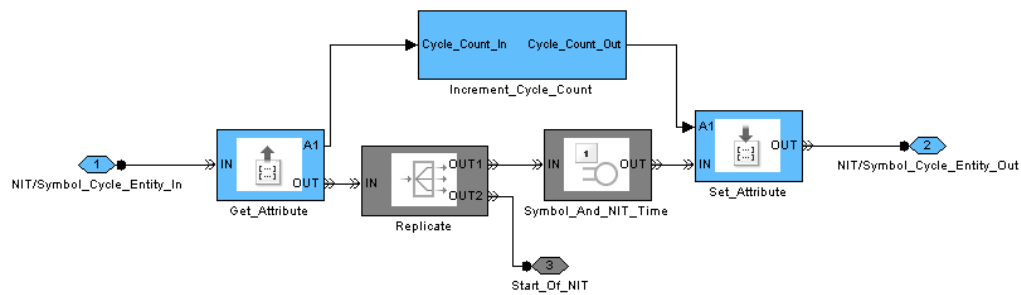


Figure 11.29: Network idle time and symbol window block

The ‘Increment_Cycle_Count’ subsystem was constructed as shown in Figure 11.30. It essentially carries out the following logic, written as ‘C’ code:

```

Cycle_count = Cycle_count + 1;
If (Cycle_count > Cycle_count_max)
    Cycle_count = 0;
End if
    
```

MODEL DEVELOPMENT

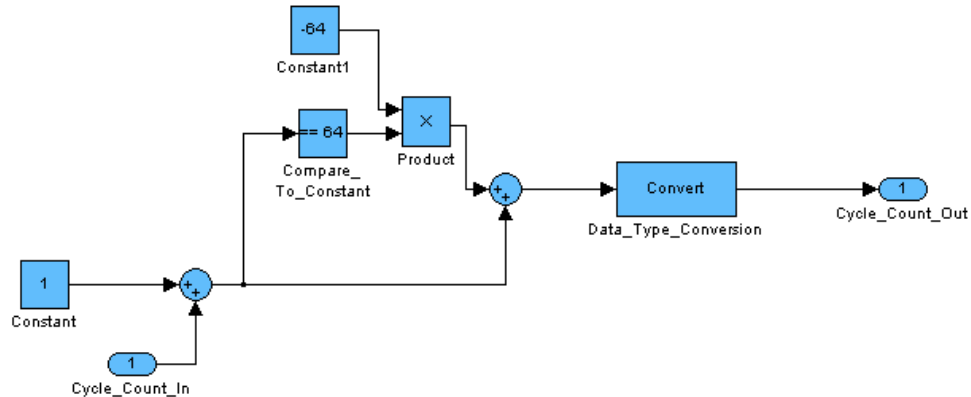


Figure 11.30: Increment cycle count block

It is important to include the non-SimEvents blocks in a ‘Discrete Event Subsystem Block’. This block helps to maintain the timing accuracy of the model. If these blocks weren’t used then false values could be taken for any calculated values. These include the amount of time an entity stays in a server for instance. This is highlighted in the help file of the SimEvents program under the heading ‘Role of Discrete Event Subsystems in SimEvents Models’ (The MathWorks, Inc. 2007):

‘The purpose of a discrete event subsystem is to call the blocks in the subsystem at the exact time of each qualifying event and not at times suggested by the time-based simulation clock. This is an important change in the semantics of the model, not merely an optimization.’

Discrete event subsystems however cannot hold SimEvents blocks. If a set of blocks makes up a subsystem of the modelled system a standard Simulink subsystem can be used to create a clear separation of components.

11.5.1.2 Global Time Unit Model

Within the model there is a ‘Synchronisation’ subsystem and a ‘Global_Time_Unit’ subsystem. These subsystems work together to define the start of both static slots and dynamic slots. The synchronisation subsystem as discussed in 11.5.1.1, simply produces entities that define the start of static slots and minislots. However dynamic slots can overlap a number of minislots. It is therefore necessary to

MODEL DEVELOPMENT

have a separate global time unit subsystem. This is seen in Figure 11.31 and is consistent with operation of FlexRay as outlined in the protocol.

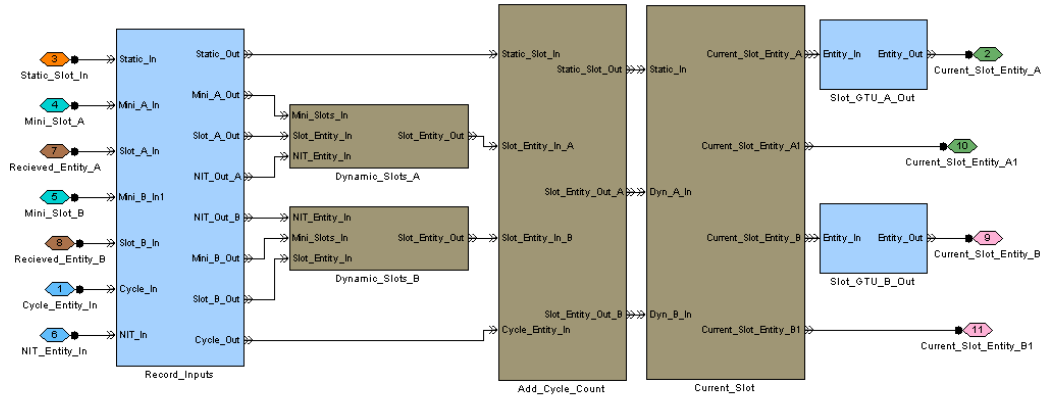


Figure 11.31: Global Time Unit

To accurately reflect the beginning of a static slot, the static slots are sent directly through the ‘Global_Time_Unit’ block. When this happens the cycle count is added as an attribute and the slot passes through to both the physical layer and to other blocks of the communications controller. This means that no delay should be experienced by a static slot. The cycle count attribute is added for uses by other layers that may be using the entity. The subsystem to add the cycle count is shown in Figure 11.32. This subsystem simple takes the cycle count from the current cycle count entity and adds it to the slots using a ‘set attribute’ block.

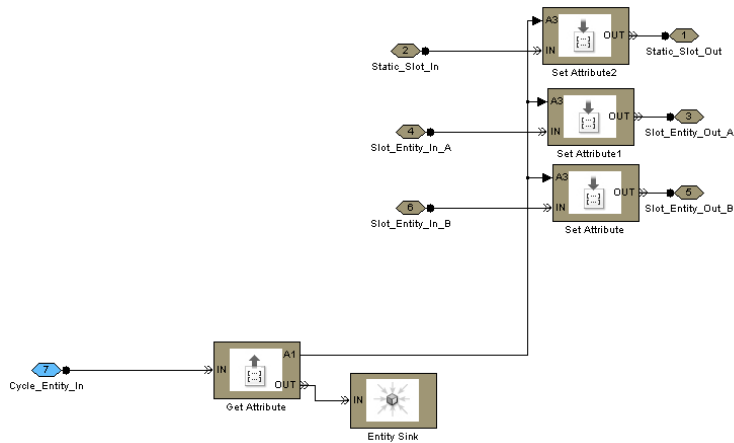


Figure 11.32: Cycle count attribute adder

MODEL DEVELOPMENT

In order to accurately reflect the beginning of a dynamic slot however is more complicated. In order to do this a slot must be taken back from the physical bus model. When a static slot arrives back from the bus its slot number is checked. If it is any slot besides the last static slot it is discarded. If it is the last static slot, the slot number is incremented and it is sent out. It will only be sent out however when a minislot arrives. This relies on the physical bus subsystem being able to hold the slot while a message is on the bus. If this does not happen, a new dynamic slot will be generated every time a minislot arrives. Figure 11.33 shows the basic components of the dynamic slot generator subsection of the 'Global_Time_Unit' subsystem.

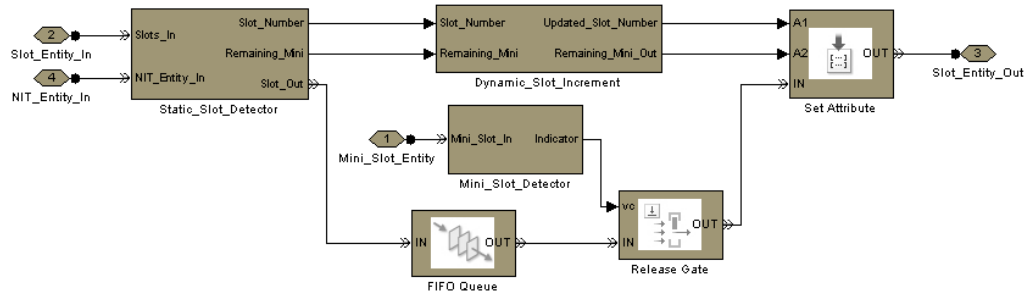


Figure 11.33: Dynamic slot generator

This subsystem also uses a copy of the cycle entity from the 'Symbol_Window_And_NIT' from the synchronisation subsystem to indicate that no more dynamic slots should be produced. Any pending entity can then be discarded.

11.5.1.3 Message Handler and Message RAM

Figure 11.11 shows the breakdown of an E-Ray communications controller. It shows the various components necessary to achieve a FlexRay compliant communications controller. In Figure 11.34 below a small section of the diagram is shown. It describes all the connections to the Message Handler and the Message RAM. As can be seen only the Message Handler has access to the Message RAM. Other systems can only access the Message RAM through the Message Handler. This should prevent any access conflicts to the RAM. Not all systems connected to the Message Handler are shown. For instance the current slot is shared by the global time unit so that any pending message can be transmitted at the appropriate time.

MODEL DEVELOPMENT

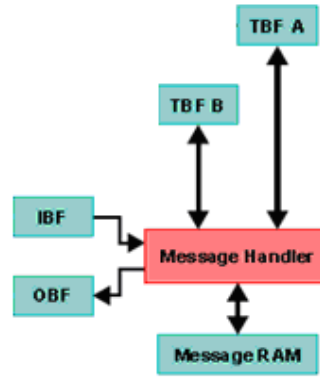


Figure 11.34: Message Handler

The basic operation of the Message Handler and Message RAM can be seen in Figure 11.35. The basic operation is as follows:

Any request for data, current slot or received data entities can be accepted by the message handler which passes them to the message RAM. The message RAM buffers are all modelled as frame entities. When any input is received the buffer entities are searched for a matching buffer to the input. If a matching buffer entity is located the buffer is updated or the value is read out to be transferred to the application layer or passed to the physical bus as appropriate. If there is no dedicated buffer then no read or update will be experienced.

While the message buffers are being read or updated no other entity is allowed to pass through the message handler. This prevents any interference from other blocks. This also follows the operation of a real-world E-Ray communications controller. If a frame is transmitted the buffer entity associated with the transmitted frame is returned to the message buffer queue. This along with the fact that received messages are stored in the same queue meant that there was a possibility that a frame could be transmitted when not appropriate. For instance the system may be set up to only transmit a particular message once. To overcome this problem a 'TX_Type' attribute was added to these entities. This attribute is only used within the message RAM subsystem. It is used to indicate whether the data associated with that message buffer should be transmitted or not. Initially all buffers are set to 'no transmission'. If a buffer is updated then this attribute must be changed. If a frame is transmitted it must still meet filtering criteria. Also this indicator is not always set to 'no transmission'. The model can be set to either 'Single Shot' mode or 'Continuous' mode. In single shot mode this indicator is reset to 'no transmission' after a message has been sent out by the message RAM. In continuous mode the attribute is not reset and the data is transmitted every valid slot.

MODEL DEVELOPMENT

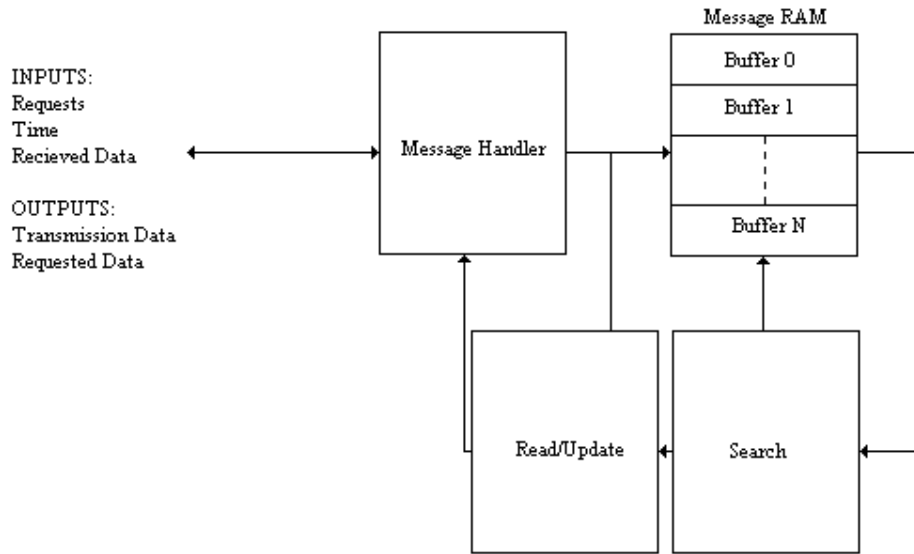


Figure 11.35: Message RAM model operation

As there is no way to directly measure the time it takes to search the message RAM and update/read a buffer (using the techniques described in Chapter 13) , there is no time associated with the search or update of the message buffers within the model. Instead the time to handle these requests is all associated with the message handler.

The message buffers are all stored within a priority queue block until a request accesses them. There can be a FIFO set up within the message buffers according to the E-Ray chip specifications. To achieve this, a separate FIFO queue block was added to the message RAM section. This does not need to be searched and as such is read and written to differently. Figures 11.36 and 11.37 show the division of the message RAM buffers and the connections to the message handler. Note how the input and output sides of the message handler are separate. This means that a busy or enable signal must be used to indicate that no more entities can enter the message handler.

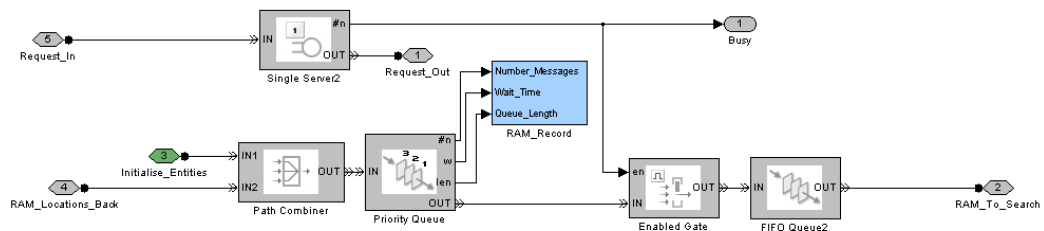


Figure 11.36: Message RAM buffers

MODEL DEVELOPMENT

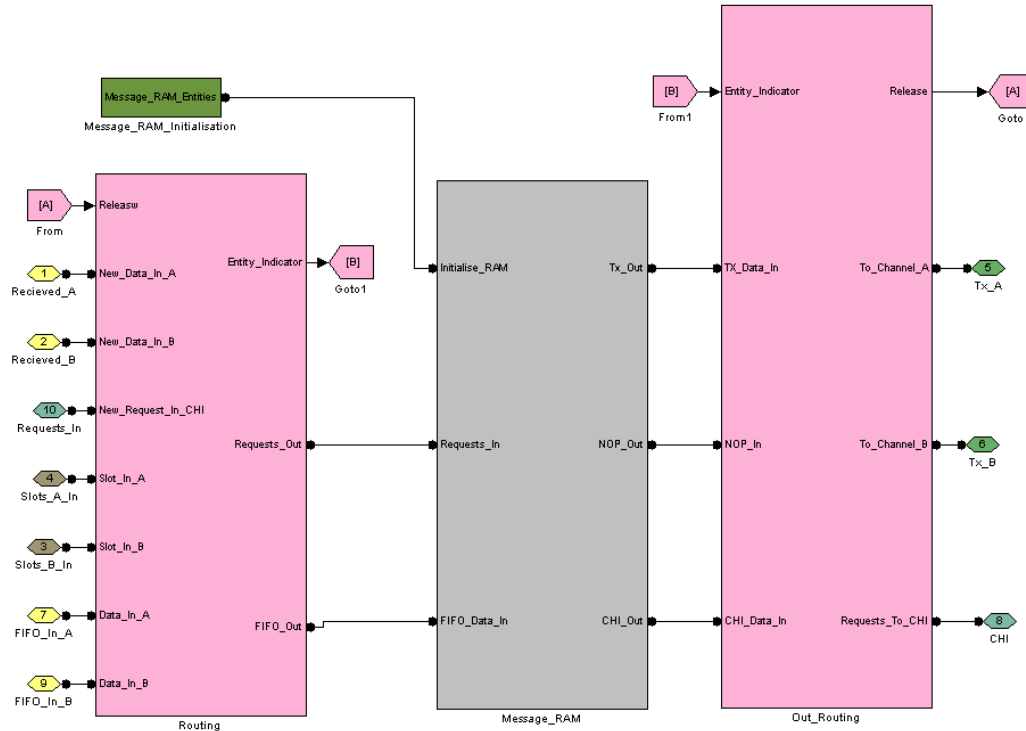


Figure 11.37: Message handler model blocks

If a request entity is received by the message handler and it is allowed to pass to the message RAM, then the requested buffer will be sent to the ‘Controller_Host_Interface’ after a calculated time based on the size of the data to be transferred. If data is to be sent out over the communication bus however it must first match given criteria. This includes cycle filter and only slots matching the appropriate slot will be sent out. When the frame is sent to the ‘Media_Access_Control’ block for transmission, it will be routed to either a the channel A or channel B access block, or both, depending on the value of the entities’ ‘Channel’ attribute.

11.5.1.4 Remaining Communications Controller Elements

The remaining communication controller elements include the ‘Media_Access_Control’, ‘Frame_And_Symbol_Processing’ and ‘Controller_Host_Interface’ subsystems. For these blocks the arriving entities will have a ‘Data’ attribute associated with them. This attribute indicates the length of data to be transferred through the subsystem and based on this value the entity will be delayed for a calculated time.

MODEL DEVELOPMENT

The ‘Media_Access_Control’ and ‘Controller_Host_Interface’ subsystems merely delay the entities; however the ‘Frame_And_Symbol_Processing’ subsystem has more functionality. This block checks arriving frames and filters them. It will reject the arriving frame entities based on the value of the ‘Frame_ID’ and what channel it arrives on. It also can be configured to filter the frames based on the current cycle. Figure 11.38 shows how the output buffer is constructed to reflect the real world implementation. As an entity enters the buffer its data length is checked and the delay it would experience due to this is calculated.

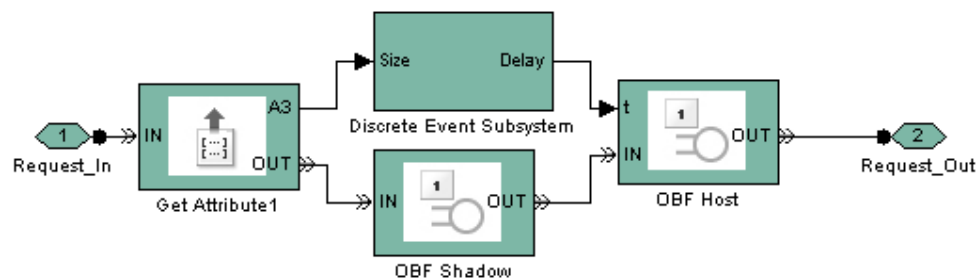


Figure 11.38: Output buffer structure

11.5.2 Physical Bus Model

The FlexRay physical bus is the medium over which data is transmitted. It consists of one or two channels, Channel A and Channel B, over which data can be transmitted. Each node is connected to one or both of the channels. There are various connection possibilities available such as active or passive star and linear bus. However a node does not know how it is connected to a FlexRay network. It only sends data over the network during its allocated slot(s). Transmission of these frames also occurs over predetermined channels. In this way any particular node is able transmit and receive data.

Figure 11.39 shows the basic operation of the physical bus. This layer must accept the slot entities generated by the simulated node. If there is a frame generated by the node the physical bus must also accept this. If there is no frame assigned to the node for a given slot, the physical bus has the potential to generate a frame entity. The physical bus may produce a frame for either both channels or just one of the channels.

MODEL DEVELOPMENT

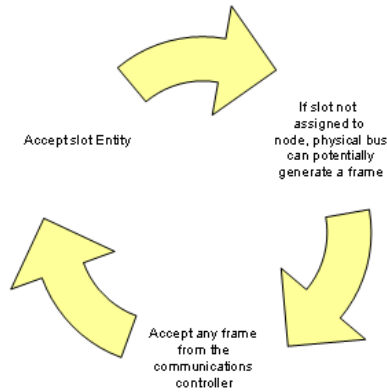


Figure 11.39: Physical operation diagram

The model of the physical bus must simulate the real world behaviour. To do this it must accept frames from the node model and generate frames to be passed to the node model. The node model will then have to determine if the frames are to be accepted or rejected. Another consideration of the physical bus is the propagation delay of the data. The bus can operate at different communication speeds. This means a 10Mbit/s data rate equates to a bit time of $0.1\mu\text{s}$, while other data rates will have other bit times. The propagation delay will be based on the data rate and length of the data to be sent. This was considered when the model was built. The main sections of the physical bus model can be seen in Figure 11.40 below.

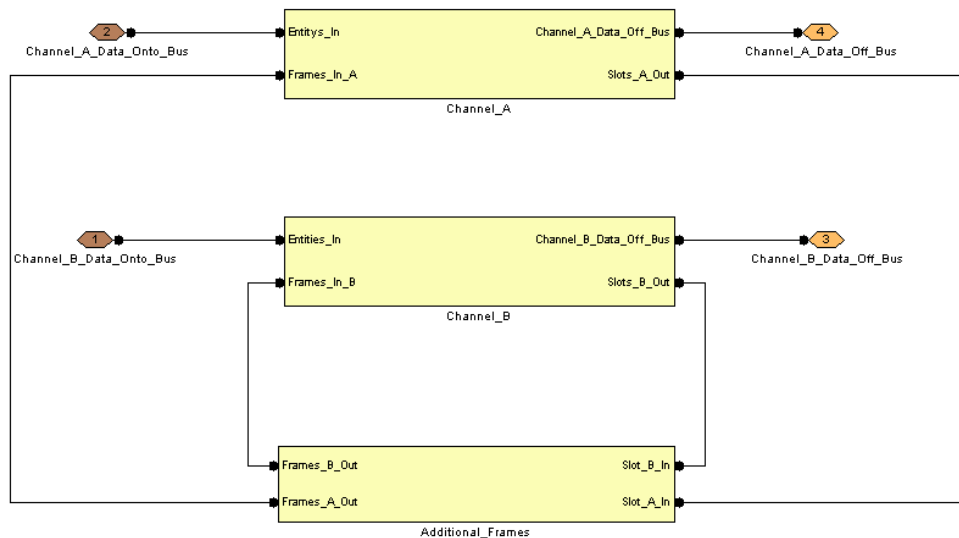


Figure 11.40: Physical bus model

MODEL DEVELOPMENT

As can be seen from Figure 11.40 the physical bus has been split up into two channels, Channel A and Channel B. There is a third subsection to the model which produces the additional frames from other nodes. These additional frames are not generated during the slot time of slots allocated for transmission from the main model node. In any other slot other than the slots allocated to the ‘main’ node a frame generation can occur. For the dynamic slots the message should only be allowed to be sent over the bus if there is sufficient time to transmit the data completely.

Figure 11.41 below shows the Channel A model. It consists of an enable gate, slot filter, a propagation delay subsystem, routing system and a record subsystem. This is the same configuration as used for Channel B.

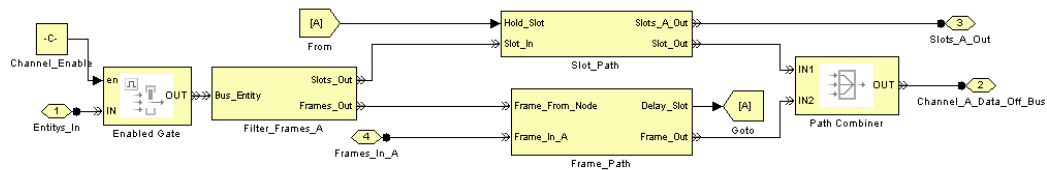


Figure 11.41: Channel ‘X’ layer

The enable gate is used to allow flexibility in test of different scenarios. Independent enabling of the channels is done using two work space variables. This allows for simulation of different node configurations such as a node connected to both or only one channel. This is consistent with real world system configuration possibilities. The frame filter block is the same subsystem as used in the ‘frame and symbol processing’ subsystem in the communications controller model block. The slot entities that are filtered in this block are sent to the alternate frames subsystem of the physical bus model as well as sent back up to the communications controller. The frames however are sent to the propagation delay block.

The elements of the propagations delay block can be seen in Figure 11.42. Figure 11.43 shows the slot routing subsystem. To calculate the propagation delay for the frames it was first necessary to obtain the data length of the transmitting frame. When this is done the server ‘Channel_’X’_Transmission_Delay’ is set to delay the frame by a calculated amount. In the slot delay block a slot will be delayed until an action point. If no frame is present on the bus after this the slot may advance from this subsystem.

MODEL DEVELOPMENT

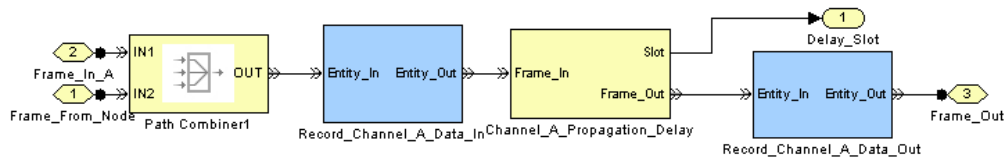


Figure 11.42: Propagation delay calculation blocks

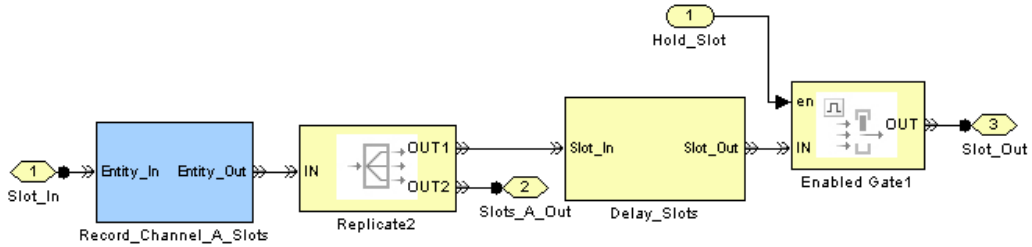


Figure 11.43: Delay slots blocks

When the frame is released from the propagation delay block it is passed with the slot entity and sent to the communications controller. In Figure 11.42 a delay slot indicator can be seen. This allows the bus to delay the reception of the slot entities by the node and this is used to define the dynamic slots. Figure 11.50 will show the same configuration as used in the delay calculation in the propagation delay calculation block. The alternative frames block is the third main block in the physical bus model. It is used to generate the additional frames that are present in a real world system. This can be seen in Figure 11.44.

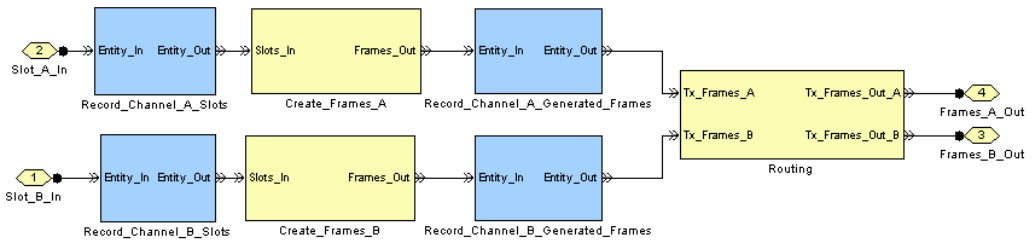


Figure 11.44: Additional frames layer

As frames can sent over both channels at the same time in static segment, Channel A sends the static slots for both channels in this case. This will help to reduce model configuration. This also means that a 'Transmit_Frame' and a 'Transmit?' block

MODEL DEVELOPMENT

for each channel are needed. These two blocks will also need to act in a different way as the transmission pattern for each channel can be different. This should not cause any problems when simulating a node connected to Channel B only. This is due to how the channel is determined using attributes. Any node connected to one channel only should have their frame attributes modified accordingly. It should also be possible to set up the system to allow different frames to be sent out on the bus during the static segment to simulate slot multiplexing.

The routing subsystem elements can be seen below in Figure 11.45. In this system the incoming frames from the previous stages are checked for the channel attribute. If this is set to just one channel the message is routed to that channel. However Channel A can generate frames that are transmitted on both channels. Therefore the frame in that case must be replicated and transmitted to both channels. This is a similar data routing subsystem as used in the message RAM block. It should be noted that channel B could alternatively be setup to transmit on both channels.

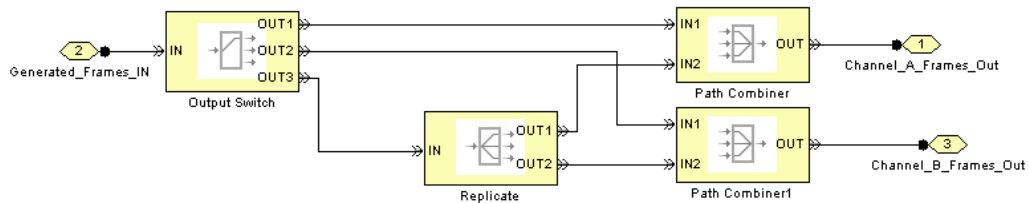


Figure 11.45: Frame routing block

11.5.3 Application Model

The application layer of the model is based on the operation of the sample programs provided for the Fujitsu SK-91F467-FlexRay evaluation boards. The particular software was the 91460_dynamic1_91467d-v16 project developed by Fujitsu Microelectronics Europe (2008). This software attempts to remain synchronised with the communication cycle. It then proceeds to run a task at the start of the communication cycle. This task sends new data, if any, to the communications controller to be transmitted over the communication network. It also requests data that may have been received and successfully stored in the message RAM. Figure 11.46 shows the flow of the application layer's basic operation. Figure 11.47 shows the application subsystem components to model this behaviour.

MODEL DEVELOPMENT

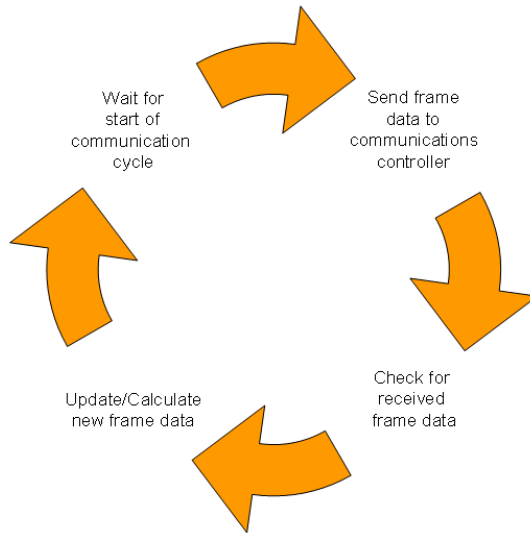


Figure 11.46: Application layer operation

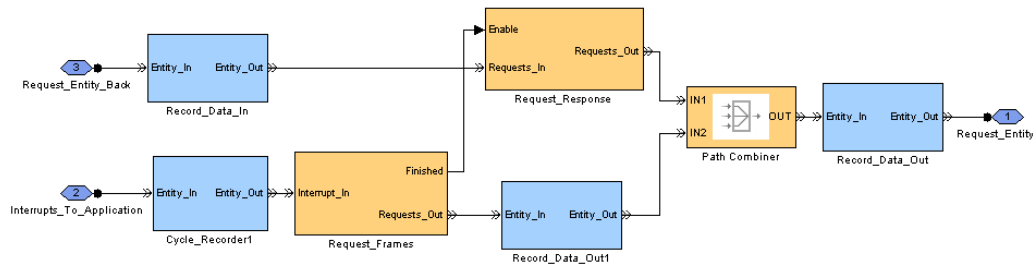


Figure 11.47: Application layer

The application model operates in the following way. At the start of the communication cycle an interrupt is received from the communications controller. The application then proceeds to produce new ‘request entities’. These requests can be requests to update a message buffer for transmission or to request data stored in the message RAM. Each of these requests will take a set amount of time to ‘process’ the data.

There is an additional functional subsystem included with the application model. This allows received frames to be ‘responded’ to. This means that for instance if a message is received during slot 6 the application may process this data and transmit new data during say slot 10 for example. This can be easily configured to produce any response required.

11.5.4 Software Driver Model

The software driver model is based on the DECOMSYS COMMSTACK<FLEXRAY> version 1.8. Based on a conversation with an employee of Elektrobit Corporation (DECOMSYS was bought by Elektrobit) who had worked on developing the software driver, it was discovered that there was no buffering used in the implementation of the software (Elektrobit Corporation 2008). This meant that the software driver would simply delay the data being passed to the communications controller from the host or vice versa. This delay would be based on the size of the data to be transferred in either direction. Figure 11.48 shows how the software driver has two tasks to perform. The first task is to transfer data from the host to the communications controller. The second task is to transfer data from the communications controller. The software driver can stay in one state indefinitely but can never process data flowing in opposite directions at the same time. Figure 11.49 shows the software driver subsystem.

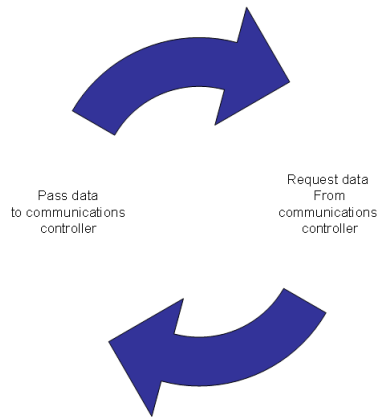


Figure 11.48: Driver operation

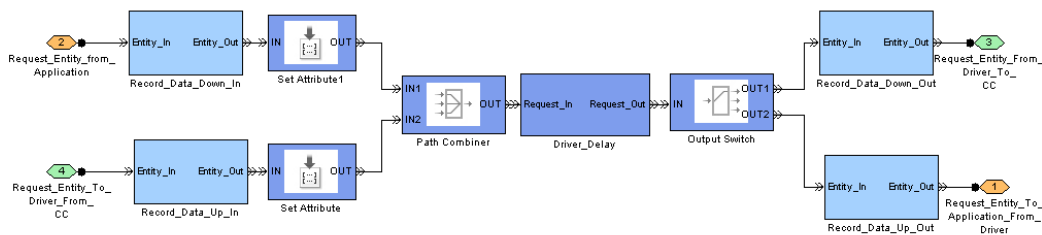


Figure 11.49: Software driver layer

It can be seen that there is only one route that entities can pass through. This allows data to flow in both directions. This implementation mirrors that of a software

driver. A software driver is a bit of code that can usually only handle one task at a time. To ensure the entities are routed correctly an attribute (Routing) is set as they enter the subsystem. This attribute is then read by the routing block which passes the data in the right direction.

Figure 11.50 shows the setup to calculate the delay of the software driver. The discrete event subsystem is used to calculate the delay. Based on the SimEvents help there is a single server block placed between the get attribute block and the server that carries out the delay. The ‘buffer’ server is set to a delay of zero. This ‘double server’ setup is used where ever calculations are done using a discrete event subsystem.

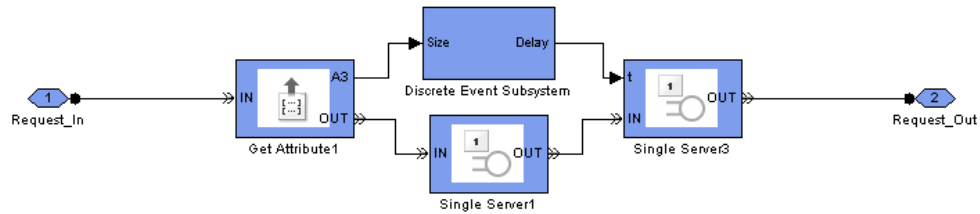


Figure 11.50: Software driver delay

11.5.5 Data Recorder Subsections

All the ‘baby blue’ coloured blocks in the model are used to save data to the MATLAB workspace. A list of all variables that can be saved can be found in Appendix B. Figure 11.51 below shows a recorder taken from the physical bus. This recorder sends the transmission frames attribute values to the workspace. This is for later analysis if desired and can be enabled or disabled using a workspace variable to route the frames into an entity sink. Figure 11.52 shows how an entity passes into the record block. As an entity passes through the top layer of a recorder block it is copied and sent to the record block. The other copy of the entity is sent on to the next part of the model. As this does not contain any server block it will not affect the timing of the entity passing through the model.

MODEL DEVELOPMENT

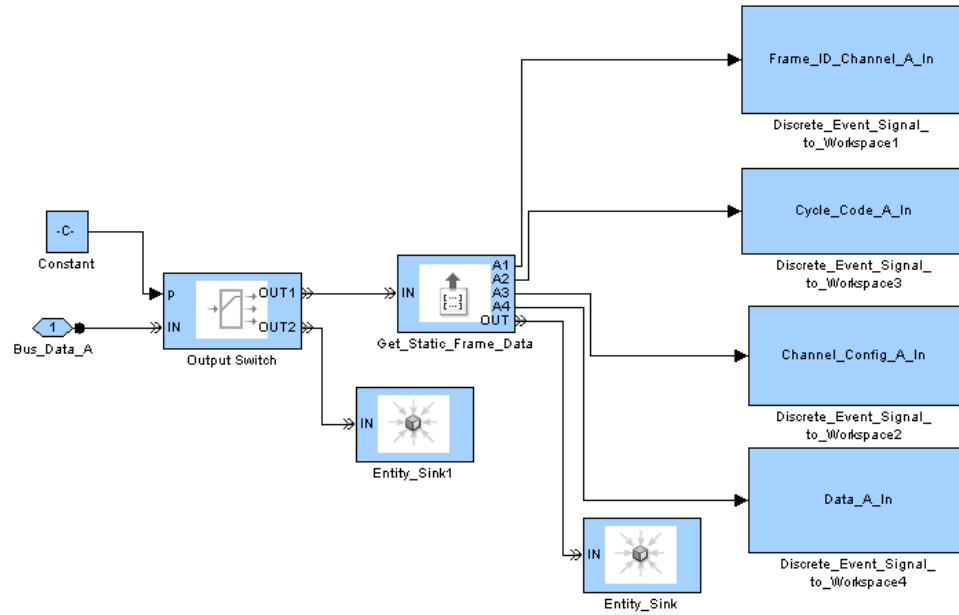


Figure 11.51: Bus monitor model

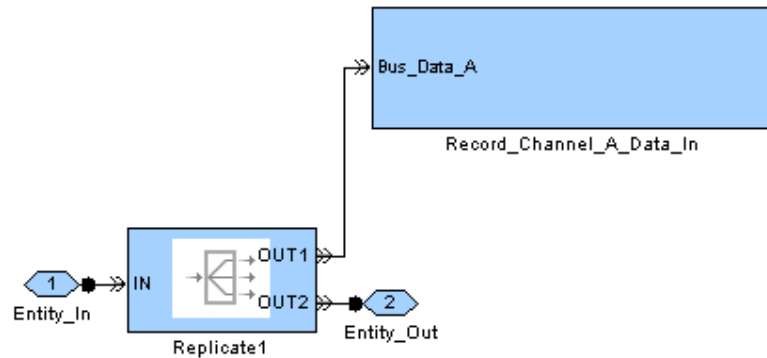


Figure 11.52: Bus monitor model

There are essentially three types of entity that may be of interest to an analyst. These are Slots, Frames and Requests. Other variations on this are the static slot, minislot, NIT and cycle start indicator or buffer entities. All these entity types are all slight variations on the slot entity and have similar recorders.

As was stated all these recorder elements can be enabled or disabled. This allows an analyst to only record data specific to the areas of interest to them. This helps reduce the amount of data stored in memory. This can also help to reduce the execution time of the model.

11.6 Conclusion

By following the specifications of the different components an accurate model was created. However not all aspects of the real world components had to be modelled. This was due to the nature and interests of the investigation being undertaken. This means that a faster executing model can be built and that the development time of the model can be kept to a minimum.

The model was viewed as a flow of data or messages from the physical bus up through the E-Ray chip and into the application through the COMMSTACK software driver. This flow of data also happens in the opposite direction, where the application sends out data through the software driver and communications controller onto the communications bus. This view of the model eased the development process and allowed the model to be broken down into functional blocks. The function of these blocks was then defined by the real-world component functionality.

Breaking down the model into different subsections also makes it more adaptable. For instance the application layer responds to an interrupt indicating the start of the communication cycle. If a user is testing a system configuration that behaves in a different way, then this model layer can simply be changed without developing a whole new overall FlexRay model. The use of workspace variables also allows users to set up the constraints of the system in a MATLAB environment without having to go through all the model's layers. This makes for a more user friendly model overall.

11.7 References

Elektrobit Corporation (2008) FlexRay training, 29 – 30 October 2008, Vienna, Austria.

FlexRay Consortium (2005) FlexRay Communication System Protocol Specification, Version 2.1 Revision A, Stuttgart: FlexRay Consortium GbR.

Fujitsu Microelectronics Europe (2008) Development Tools: SK-91F467-FLEXRAY : Fujitsu EMEA [online], available at: http://mcu.emea.fujitsu.com/mcu_tool/detail/SK-91F467-FLEXRAY.htm#SOFTWARE [accessed 19 November 2008].

Mathworks Automotive Advisory Board (MAAB) (2007) Control Algorithm Modeling Guidelines Using MATLAB, Simulink, and Stateflow Version 2.0.

MODEL DEVELOPMENT

The MathWorks, Inc. (2007) SimEvents Help, Massachusetts: The MathWorks, Inc.

Chapter 12 . Verification

12.1 Introduction

Figure 12.1 shows where the verification stages fits into the model building section. The verification stage is highlighted in Figure 12.1. The need for verification of a model was discussed in section 7.5 of this thesis. Section 7.5 also discussed the basic methodology behind the verification process. This chapter will discuss the different tests that were carried out to achieve the verification of the FlexRay simulation model.

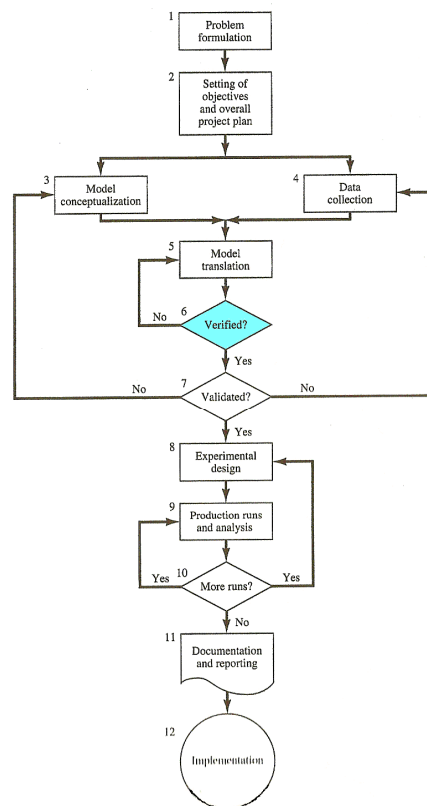


Figure 12.1: Model development flow chart

The verification process is broken down into two different sections. The first section describes the verification process used for each major model subsystem. The second section describes the method to verify the FlexRay model’s operation as a whole.

The relevant test parameters are highlighted and the data obtained from the tests documented.

12.2 Verification

E.W. Dijkstra is summarised in Francez (1992, p1) as saying “testing can reveal the presence of errors, not their absence”. From this quote it can be seen that the testing of a system can potentially be a long and complicated one. There must be enough testing performed on a system to allow the developer to certify their program is working as intended. An infinite number of test runs are impossible and impractical for a tester to perform. A small, finite number of tests may merely ‘debug’ the program as stated by Francez (1992, p1). This would mean that the system performs accurately for a given set of inputs and that every line of code executes without any errors.

Using a limited number of tests does not convey whether a program performs as intended. For example a simple function written in ‘C’ may accept a variable and scale this value by multiplying the value by a given number, say 250. If the function is not passed the expected type of variable, for instance, the function will not execute and the program needs to be debugged. During this process the programmer should fix the error and ensure either 1) the function is passed the correct type of variable or, 2) the function can handle different variable types.

Verification of a system is however distinct from the debugging phase of the system. In the verification process the system is passed a set of known parameters. The system is then executed and the output checked against the expected output of the system. In the example above it can be seen that the function described can only be verified if the output is a correctly scaled version of the input. All the functions of a system may be verified individually from the system as a whole. When all functions are integrated into an entire system, verification of the whole system may also be done.

Banks et. al. (2001, pp369- 370) describe steps that should be followed, if possible, to accurately verify a simulation model. The following steps are described as ‘common sense suggestions’ and ‘are basically the same ones any software engineer should follow’. The steps are as follows:

1. Have the model checked by someone other than the developer. They will be able to verify the model logic if the development has been properly documented.

MODEL DEVELOPMENT

2. A flow diagram of the possible actions a system can take when event occurs should be developed. All these possibilities can then be checked for correct performance.
3. Check the output of the model for 'reasonableness' for a wide range of inputs.
4. Make sure that the input parameters are correct. This should be done at the start and end of the tests. This ensures that the values obtained are a match for the inputs the developer wished to check.
5. Document the operation of each major section of the model. Also document the use and definition of every variable used in the system.
6. If a visual output from the model can be obtained, ensure that this output also accurately represents the expected output. This should also be used during the testing phase if possible even if the final implementation of the model does not produce any visual output.
7. Make use of any debugging functionality available to monitor the program. It may be necessary to concentrate on the subsystems one at a time, but this will make the overall model more robust.

A number of these steps were followed as closely as possible during the verification process of the FlexRay simulation model. This process is described in section 12.2.1.

Some tests may produce a large amount of data that may be 'extremely cumbersome' to check for correctness (Banks et. al. 2001, p374). However it is necessary to check that all possible events occur and the correct action is followed. Also a short simulation will produce a set of outputs that is easier to check. In this way artificial data could be used to produce the occurrence of all events, no matter how rare they may be (Banks et. al. 2001, p374). As this stage precedes the calibration stage the model does not have to accurately represent the timing of the actual system. In this way the system must only respond in the correct manner based on the inputs applied to the system.

12.2.1 Model Verification Procedure

The model verification was split into different two types of verification. The first was a 'debugging' procedure that was aimed at ensuring the individual model subsystems worked as intended. The tests at this stage were designed to discover any flaws in the basic operation of the models subsystems. Using the MathWorks support and the help files solutions to problems were quickly implemented (The MathWorks,

Inc. 2008). The tests were then run again until the models subsystem performed as desired.

The second stage of the verification process was to integrate the blocks and test them. This was important as problems can occur when integrating the different blocks. These problems could be simple syntax errors where different blocks expect different attribute names or a misspelling has occurred. The model at this stage was also analysed for correctness of behaviour. An example would be to check that the model would only transmit during the assigned slots.

The following sections, section 12.3 and 12.4 will outline the debugging and verification processes separately.

12.3 Model Subsystem Debugging

Each of the model subsystems is intended to perform a different task. In order to test the model each subsystem must be fully compatible with the other model subsystems. Tests were developed to test the functionality of each subsystem. When all the debug tests were run with no error the subsystem could be considered as being debugged. Section 12.3.1 will outline the methods and functions available to debug and verify the model blocks. These methods and functions are standard MATLAB or SimEvents features. These were also used throughout the integration verification, calibration and validation stages of the model building process.

Figure 12.2 shows a block diagram of each of FlexRay simulation model subsystems. These represent each of the blocks that were constructed and tested as described in this chapter. In Figure 12.2 OBF stands for output buffer, IBF is the input buffer, FSP is the frame and symbol processing subsystem and MAC is the media access control subsystem. The GTU subsystem is the global time unit subsystem. This subsystem does not pass information. Instead it controls the view of the current communication slot.

The FlexRay simulation model groups the OBF and IBF blocks together into a controller host interface (CHI) subsystem. The verification of these E-Ray subsystems will therefore be discussed as one FlexRay simulation model subsystem.

MODEL DEVELOPMENT

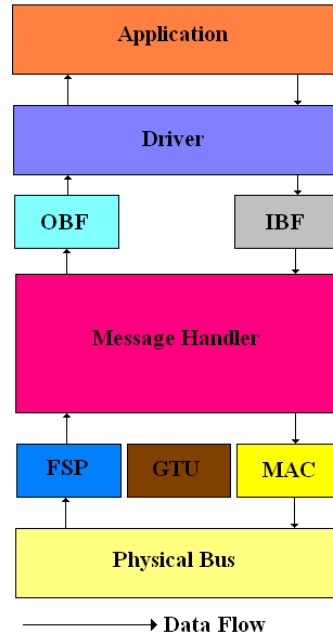


Figure 12.2: Model subsystem block diagram

To debug and verify the different subsystems a number of tests were run. These tests checked for correct execution of the subsystem. During the debugging process it was noticed that execution of a subsystem could quickly be performed. In a number of the debugging tests the problems observed involved the subsystem expecting a slight attribute name difference. This could be a simple case of mistyping the attribute during the model development stage. However the execution of the model subsystem may still be incorrect. For each model subsystem a number of tests were developed and run. An example of these tests was for the synchronisation block where a number of tests were run to observe the correct generation of the static and minislots. In a number of cases during some cycles an extra slot would pass through the enable gate of either the static section or the dynamic section. The model was changed to produce the correct output from this block. The subsystem was then considered as working as intended after all the tests produced the correct output.

The output from the model synchronisation subsystem test was obtained and is shown in Table 12.1. This was done by storing the arrival times of static slot and minislot entities at the output stage of the synchronisation model block subsystem for example. All the recorded values were stored in the MATLAB workspace for later analysis. This method was used in all tests where the behaviour of the model was

MODEL DEVELOPMENT

observed. This is a useful feature of MATLAB and SimEvents as the data can then be analysed without exporting the data to another software package such as MS Excel.

Cycle Times	Static Slots	Mini Slot A	Mini Slot B
0.000000	0.000000	0.002100	0.002100
0.005000	0.000035	0.002110	0.002110
0.010000	0.000070	0.002120	0.002120
0.015000	0.000105	0.002130	0.002130
0.020000	0.000140	0.002140	0.002140
0.025000	0.000175	0.002150	0.002150
0.030000	0.000210	0.002160	0.002160
0.035000	0.000245	0.002170	0.002170
0.040000	0.000280	0.002180	0.002180
0.045000	0.000315	0.002190	0.002190
0.050000	0.000350	0.002200	0.002200

Table 12.1: Synchronisation test 1 simulation time results

The data shown in Table 12.1 is a small portion of the data recorded during the run of test case 1. The time for each cycle, static slot and mini slot was then analysed for correctness. This is shown in Table 12.2.

Time between cycle starts	Static slot length	Mini slot length A	Mini slot length B
0.005000	0.000035	0.000010	0.000010

Table 12.2: Synchronisation test 1 results summary

Graphs were also obtained of the arrival entities at a given point of the model. These graphs were used to check that any entity did not arrive out of sync. They could also be used to display entity attributes as desired. This can be seen below in the synchronisation test 1 cycle entity attribute scope, Figure 12.3. Note that there should be 2 rounds of 64 cycles = 128 cycle entities. However there are 129 entities in the diagram below. This is because the entity is detected just at the end of the simulation time. The behaviour is the desired output from the model.

MODEL DEVELOPMENT

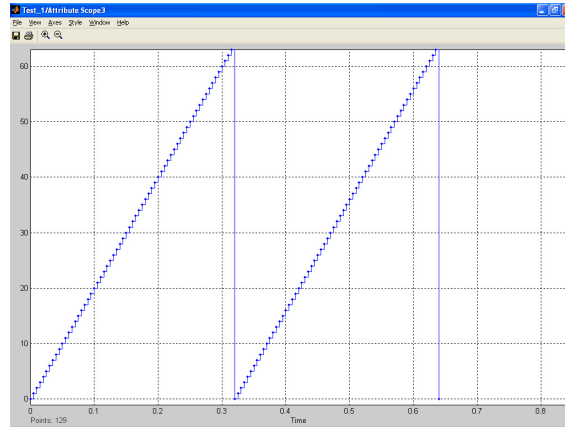


Figure 12.3: Synchronisation test 1 attribute scope graph for cycle entities

As can be seen from the sample data and test summary tables given above, the model produces data and this can be easily analysed in the MATLAB environment. These data capture and analysis techniques for observing the models behaviour have proven to be sufficient to carryout the testing procedures. These techniques were therefore used to observe all models behaviour including the calibration and validation tests as described in chapters 13 and 14.

During a number of the integration debugging tests the model did not work as intended. In many cases the model would end up in an infinite loop. This would mean that the simulation time would not advance and the simulation run would need to be forcibly stopped. The problem was traced back to the message handler model subsystem. When this block was being tested it was necessary to increase the total number of debug tests for this subsystem. A wider variety of tests were necessary to ensure that every aspect of the message RAM worked. The tests in some cases would replicate a given condition more than once. For example there were four tests carried out to check if the message handler would retrieve a message stored in the RAM and pass the information to the host.

During the frame and symbol processing subsystem testing a number of problems were dealt with. An example of this is that an initial frame that entered the block would be accepted for storage in the message RAM. This was even the case when the subsystem was set up to reject the entity. The filtering for the remaining frames would also be one frame off. For example frame 5 would be accepted instead of frame 4.

During the testing of the software driver subsystem a number of problems were observed. These included the blocking of information passing through the software

driver in one direction. The software driver model was adjusted until the desired performance was obtained. The software driver was then considered verified.

When the application layer tests were run a number of errors were detected. These included the incorrect setting of attributes. The model was then modified to fix any problems. Each of the tests was run until the desired output was achieved. Each test of the verification process eventually produced the correct results. This meant that the model was performing as desired; therefore the verification of the application layer was deemed a success.

12.3.1 Model Systems Debugging Results

Each of the main model subsystems were broken down and tested as they were built. As problems were discovered the model subsystem was changed until the correct performance was observed. The debugging tests allowed the system to be tested for bugs as well as correct performance. This should ensure correct execution of the model in all situations. The model can then be verified as a whole and the correct performance of the subsystems should be observed. The methods to obtain and analyse data from the model have also been tested as sufficient to carry out the verification procedure.

As the verification stage happens before the calibration stage, there is no way to test the delay calculation blocks completely. These blocks must be developed after the calibration data is collected. Only when the calibration data is collected is the relationship between the length of time a block takes to execute and a given variable, for instance payload size, known.

12.4 Simulation Model Verification

The following test case parameters, as listed in Table 12.3, were developed to verify the FlexRay simulation model. The configuration constraints listed in Appendix B of the FlexRay specification were consulted for relevant parameters when developing the Test Cases (FlexRay Consortium 2005, pp.214-220). Only parameters that were relevant to the study were considered.

Other constraints that needed to be considered were the limitations of the tools available to configure the FlexRay settings. These tools are discussed in chapter 13. The tools limit the number of buffers that can be assigned for instance. This impacts the

MODEL DEVELOPMENT

calibration and validation constraints and limits the number of messages that can be assigned. As the tools impact the real-world system setting it will impact the simulation model settings.

Table 12.2^{**} illustrates the parameters used in the verification stage of model development process. The test case parameters were designed to test the functionality of the system. These tests are still useful to test to see if the messages are transmitted or received correctly.

^{**} The BMW example timing is taken from an article by Berwanger et. al. (2004, pp. 6-8). This article highlights how BMW intend to have a fixed communications schedule with a static segment with a length of 3ms and a dynamic segment of 2ms approximately. This has been followed as closely as possible. In the table, any letter in brackets following a frame ID refers to the communication channel a frame is transmitted/received on. If there is only one channel or the frame is transmitted on both communication channels then the frame ID is not indicated.

MODEL DEVELOPMENT

ID	Cycle Length	Number of Static Slots	Number of Mini Slots	Static Slot Length	Mini Slot Length	Static Frame Payload	Dynamic Frame Payload (max)	Channels	NIT & Symbol Length	Node Tx Frames	Node Rx Frames	Latest Tx	Note
1	16000 μ s	630	0	25 μ s	NA	1 word	NA	A&B	250 μ s	3 and 44	6 and 18	0	Max cycle length, no mini slots
2	16000 μ s	2	1548	43 μ s	10 μ s	10 words	20 words	A&B	431 μ s	2 and 444 (A)	1 and 181 (B)	1543	Max cycle length, Min static slots
3	5000 μ s	20	209	123 μ s	10 μ s	50 words	80 words	A&B	447 μ s	3 and 65 (A)	6 and 66 (B)	192	Medium cycle length/number of static slots/ number of mini slots
4	5000 μ s	17	34	278 μ s	6 μ s	127 words	5 words	A&B	67 μ s	3 and 28 (A)	6 and 29 (B)	29	Large static slot size, small number mini slots
5	114 μ s	2	0	27 μ s	NA	2 words	NA	A&B	60 μ s	2	1	0	Min cycle length
6	5000 μ s	60	239	36 μ s	10 μ s	6 words	20 words	A&B	447 μ s	3 and 65	8 – 15 (all A)	234	received frames stored in FIFO
7	5000 μ s	60	276	35 μ s	10 μ s	1 word	16 words	A&B	137	3 and 65 (A)	6 and 66 (B)	271	Based on the CANalyzer example
8	5000 μ s	79	148	31 μ s	10 μ s	8 words	16 words	A&B	436	3, 10, 52 and 159 (A)	6 and 155 (B)	143	Based on the BMW example

MODEL DEVELOPMENT

9	5000 μ s	60	245	35 μ s	10 μ s	4 words	20 words	A	447 μ s	3 and 65	6 and 66	240	Channel A only
10	9908 μ s	120	490	35 μ s	10 μ s	4 words	20 words	B	805 μ s	3 and 65	6 and 66	485	Channel B only, max NIT
11	300 μ s	6	12	31 μ s	6 μ s	2 words	20 words	A&B	63 μ s	3 and 7 (A)	6 and 8 (B)	2	Small static slot and mini slot
12	4354 μ s	60	195	39 μ s	10 μ s	8 words	16 words	A&B	61 μ s	3 and 65 (A)	6 and 66 (B)	190	Min NIT/Symbol window.
13	15982 μ s	2	2640	39 μ s	6 μ s	8 words	60 words	A&B	61 μ s	2 and 100 (A)	1 and 770 (B)	2617	Max number of mini slots and min static slots /NIT and Symbol window
14	5408 μ s	8	0	659 μ s	NA	127 words	NA	A&B	136 μ s	3	6	0	Max static slot length and payload

Table 12.3: Verification test case parameters

MODEL DEVELOPMENT

Once the testing parameters were defined, it was necessary to define the desired output of the system. Each of the test cases were analysed and the desired performance of the model was determined. In each case it was determined that for each test case there should be at least one request generated from the application layer and passed through the software driver to the communications controller. These requests could be to indicate a desire to transmit data, or a request for received data. Therefore it would also be necessary to see what messages were stored in the message RAM as well as what data was transmitted from the node. If the model performed in the desired way then the model was deemed as being verified as performing correctly.

12.4.1 Simulation Random Numbers

The simulation model makes use of random number generators to vary the generated frames by the physical bus layer model. By using seeds the results obtained from a single simulation run are repeatable. To obtain a different set of random numbers, different seeds must be used. A seed value must be a number between 0 and 4,294,967,295 (The MathWorks, Inc. 2008). Table 12.4 shows the random numbers that were used for each of the 14 test cases.

Test Case	Application Generation	Application Response	Physical Bus A-1	Physical Bus A-2	Physical Bus B-1	Physical Bus B-2
1	901	1763	272	777	15973	12
2	231	405706	19881	3103	568	86418
3	6068	93	1527392702	790176266	708	6382325
4	860	916904	4000676564	72711	411327	284444
5	891	270206	432287947	159790	93808	1001
6	7621	8	1578461665	9797	173	63272
7	456511	578	1675424	9414884	7	2214
8	185	68132217	7506	56238	200926	205738
9	821433	303758	21417824	216	95413	445253
10	4447	660924	24963	6602275	25548	353118
11	6154	1388908	57650387	39245	72912	5971
12	79	21856	67	384193844	1829106	465256
13	9218	7426	505	60735	377	8074
14	732	938	864818	516961	632116	87399

Table 12.4: Verification test case random number seeds

MODEL DEVELOPMENT

This method of using seeds helps create repeatable simulation runs and therefore repeatable results.

12.4.2 Verification Test Cases and Results

The verification test cases were based on the verification test case parameters. The following table, Table 12.3, describe the expected behaviour of the model during these tests. For each layer an expected behaviour is given. If the simulation model performed as desired the result was a pass. If the model did not produce the correct response the result was a fail. Tables 12.5-12.18 summarises the verification test cases and verification result.

Test Case	Application	Driver	Communications Controller	Physical Bus
1	Send an update for frames 3 and 44. Request data received from frames 6 and 18	Handle requests in an appropriate manner	Update relevant buffers based on data received from either the physical bus layer or application layer. Transmit frames 3 and 44 correctly	Generate a set of frame entities
Result: Pass				

Table 12.5: Verification test case 1 result summary

Test Case	Application	Driver	Communications Controller	Physical Bus
2	Send an update for frames 3 and 181. Request data received from frames 6 and 444	Handle requests in an appropriate manner	Update relevant buffers based on data received from either the physical bus layer or application layer. Transmit frames 3 and 181 correctly	Generate a set of frame entities
Result: Pass				

Table 12.6: Verification test case 2 result summary

MODEL DEVELOPMENT

Test Case	Application	Driver	Communications Controller	Physical Bus
3	Send an update for frame 3. Request data received from frames 6 and 66. Then generate an update for frame 65	Handle requests in an appropriate manner	Update relevant buffers based on data received from either the physical bus layer or application layer. Transmit frames 3 and 65 correctly	Generate a set of frame entities
Result: Pass				

Table 12.7: Verification test case 3 result summary

Test Case	Application	Driver	Communications Controller	Physical Bus
4	Send an update for frames 3 and 28. Request data received from frames 6 and 29	Handle requests in an appropriate manner	Update relevant buffers based on data received from either the physical bus layer or application layer. Transmit frames 3 and 28 correctly	Generate a set of frame entities
Result: Pass				

Table 12.8: Verification test case 4 result summary

Test Case	Application	Driver	Communications Controller	Physical Bus
5	Send an update for frame 2. Request data received from frame 1	Handle requests in an appropriate manner	Update relevant buffers based on data received from either the physical bus layer or application layer. Transmit frame 2 correctly	Generate a set of frame entities
Result: Pass				

Table 12.9: Verification test case 5 result summary

MODEL DEVELOPMENT

Test Case	Application	Driver	Communications Controller	Physical Bus
6	Send an update for frames 3 and 65. Request three frames stored in the FIFO	Handle requests in an appropriate manner	Update relevant buffers based on data received from either the physical bus layer or application layer. Transmit frames 3 and 65 correctly	Generate a set of frame entities
Result: Pass				

Table 12.10: Verification test case 6 result summary

Test Case	Application	Driver	Communications Controller	Physical Bus
7	Send an update for frames 3 and 65. Request data received from frames 6 and 66	Handle requests in an appropriate manner	Update relevant buffers based on data received from either the physical bus layer or application layer. Transmit frames 3 and 65 correctly	Generate a set of frame entities
Result: Pass				

Table 12.11: Verification test case 7 result summary

Test Case	Application	Driver	Communications Controller	Physical Bus
8	Send an update for frames 3, 10, 52 and 159. Request data received from frames 6 and 155	Handle requests in an appropriate manner	Update relevant buffers based on data received from either the physical bus layer or application layer. Transmit frames 3, 10, 52 and 159 correctly	Generate a set of frame entities
Result: Pass				

Table 12.12: Verification test case 8 result summary

MODEL DEVELOPMENT

Test Case	Application	Driver	Communications Controller	Physical Bus
9	Send an update for frames 3 and 65. Request data received from frames 6 and 66	Handle requests in an appropriate manner	Update relevant buffers based on data received from either the physical bus layer or application layer. Transmit frames 3 and 65 correctly	Generate a set of frame entities
Result: Pass				

Table 12.13: Verification test case 9 result summary

Test Case	Application	Driver	Communications Controller	Physical Bus
10	Send an update for frames 3 and 125. Request data received from frames 6 and 126	Handle requests in an appropriate manner	Update relevant buffers based on data received from either the physical bus layer or application layer. Transmit frames 3 and 125 correctly	Generate a set of frame entities
Result: Pass				

Table 12.14: Verification test case 10 result summary

Test Case	Application	Driver	Communications Controller	Physical Bus
11	Send an update for frames 3 and 7. Request data received from frames 6 and 8	Handle requests in an appropriate manner	Update relevant buffers based on data received from either the physical bus layer or application layer. Transmit frames 3 and 7 correctly	Generate a set of frame entities
Result: Pass				

Table 12.15: Verification test case 11 result summary

MODEL DEVELOPMENT

Test Case	Application	Driver	Communications Controller	Physical Bus
12	Send an update for frames 3 and 65. Request data received from frames 6 and 66	Handle requests in an appropriate manner	Update relevant buffers based on data received from either the physical bus layer or application layer. Transmit frame 3 and 65 correctly	Generate a set of frame entities
Result: Pass				

Table 12.16: Verification test case 12 result summary

Test Case	Application	Driver	Communications Controller	Physical Bus
13	Send an update for frames 3 and 100. Request data received from frames 6 and 770	Handle requests in an appropriate manner	Update relevant buffers based on data received from either the physical bus layer or application layer. Transmit frames 3 and 100 correctly	Generate a set of frame entities
Result: Pass				

Table 12.17: Verification test case 13 result summary

Test Case	Application	Driver	Communications Controller	Physical Bus
14	Send an update for frame 3. Request data received from frame 6	Handle requests in an appropriate manner	Update relevant buffers based on data received from either the physical bus layer or application layer. Transmit frame 3.	Generate a set of frame entities
Result: Pass				

Table 12.18: Verification test case 14 result summary

For each of the verification test cases the model acted as desired. The models output was checked at various stages and verified as the correct response. As the model acted as desired for each of the given test cases the model can be deemed to be verified.

12.5 Model Execution Time

During the initial stages of the integration testing of the model it was noted that the model was extremely slow to execute. This time was generally 3+ hours for a single communications cycle. This time is clearly not a practical time for the system to execute to be a useable tool. A number of methods were used to attempt to speed up the simulation model. These included removing as many floating point calculations as possible and reducing the number of blocks that execute. To achieve this, the profiler tool in Simulink was used to highlight any blocks that were rarely executed. From this it was noticed that the model recorder blocks, even when switch 'off' were producing an output of all 0's. This meant that the model was outputting a lot of data to RAM despite having nothing to output.

To ensure that all the proposed solutions were producing results, a number of tests were run. These tests were conducted on two sets of testing parameters as well as combinations of system parameters.

The first test run was based on a communications cycle of 5000 μ s. This was broken into a static segment consisting of 60 static slots with duration of 35 μ s. The network idle time and symbol window had duration of 450 μ s and the dynamic segment was broken down into 245 mini slots with duration of 10 μ s each. This test essentially used the parameters as described for verification test case 7.

The second test run was based on a communications cycle of 5000 μ s. This was broken into a static segment consisting of 82 static slots with duration of 60 μ s. The network idle time and symbol window had duration of 50 μ s and the dynamic segment was broken down into 5 mini slots with duration of 6 μ s each. This test essentially used the parameters as described for verification test case 8.

The tests were then run with different configurations of system parameters set. The time (to the nearest minute) was recorded for both the start and end times of the simulation runs. The results and system parameters are summarised below in Table 12.19. It should be noted that each of the tests were run under the accelerator mode in SimEvents and that as many variables from the workspace were defined as static in the optimisation pane of the simulation setup. Originally the tests were performed on a Dell Optiplex GX270 with a Pentium 4 processor running at 2.8GHz and with 512MB of RAM. The version of MATLAB was R2006a. They were then transferred to a Dell Optiplex 745 with a Pentium D running at 3GHz and with 1GB of RAM. The version of

MODEL DEVELOPMENT

MATLAB on this machine was R2008a. All values listed in Table 12.19 were obtained from the second computer.

Test	Floats used	Profiler status	Recorder modules	Testing Parameters	Start Time	End Time	Time Taken (approx)
1	Yes	Yes	Yes	1	9.35	13.25	3 hours 50 minutes
2	No	No	Yes	1	15.08	15.30	22 minutes
3	Yes	No	No	1	14.13	14.53	40 minutes
4	No	No	No	1	15.05	15.08	5 minutes
5	No	Yes	No	1	9.47	10.10	23 minutes
6	Yes	Yes	No	1	10.10	13.53	3 hours 43 minutes
7	No	No	No	2	10.13	10.17	4 minutes
8	Yes	Yes	No	2	10.18	10.41	23 minutes
9	No	No	Yes	2	12.17	13.11	54 minutes
10	Yes	Yes	Yes	2	9.27	14.55	5 hours 28 minutes

Table 12.19: Speed tests

In Table 12.19 all the times in the ‘Time Taken (approx)’ column is colour coded. If the time taken to execute the model was deemed in the preferable time range (<10 minutes) is marked in green. Any time deemed as acceptable (10-30 minutes) is marked as orange and any other time is marked in red.

It can be seen that eliminating the analysing software reduces the execution time of the model. There is also a noticeable difference when comparing the execution time of two different FlexRay parameter configurations. Finally the elimination of any floating point calculations also helped reduce the execution time of the model. To eliminate floating point calculations all timing values were presented as nanoseconds. The analysis the output of the model must therefore take this into account. Lookup tables were also used where pre-run-time calculations could be performed.

12.6 Conclusion

During this phase of the model development, the model was purely tested for functionality. The tests were designed to reproduce as many errors as possible in a controlled environment. Each of the model sections was tested for functionality using a basic prototype. This was tested for soundness of concept before being fully implemented. This implementation was then tested and verified on its own. When this was done all the subsystems were integrated into the final model. The integrated model was then verified as a single FlexRay simulation model system.

During the integration testing, a number of errors were found. These ranged from basic errors such as mistyped attribute names and routing errors (i.e. an 'out-port' of a subsystem was not connected to the correct 'in-port' of a following subsystem), to more complicated problems. Such problems included a problem where two request entities attempted to access the message handler at the same time. The problems were dealt with one at a time and appropriate solutions found. The model was again tested for correct functionality. Only when no errors were observed and the functionality of a system observed as correct, was the FlexRay model and its subsystems deemed to be verified.

As can be seen (from the timing analysis section of the testing) a number of techniques were used to speed up the execution time of the model. The profiler tool was used as an example to demonstrate the use of a different type of analysis tool besides the recorder modules. Each of the techniques mentioned helped reduce the execution time of the model.

It can be seen that any model that uses any form of analysis tools will be slowed down. However eliminating all analysis tools produces a model that is useless as no data can be obtained from it. This means that any model should only contain the necessary recorder modules needed to carry out a study on a system. In this way it was discovered that the general-purpose model design could not be practically implemented using the available equipment. This also focuses the model from a general into a specific analysis tool. An alternative approach to a general-purpose model is to allow the user of the model to create recording modules for each separate test they wish to conduct. However this will make the model less user friendly.

12.7 References

Banks, J., Carson, J. S., Nelson, B. L. and Nicol, D. M. (2001) Discrete-Event System Simulation, New Jersey: Prentice Hall.

Berwanger, J., Schedl, A. and Peller, M (2004) BMW – First Series Cars with FlexRay in 2006, Automotive electronics + systems Special Edition, Development Solutions 19 for FlexRay ECUs, 6-8.

Clune, M. (2008) RE: Denying access to entities until a task is finished in SimEvents, email to Robert Shaw (rshaw@wit.ie), 8 December [accessed 15 Dec 2008].

FlexRay Consortium (2005) FlexRay Communication System Protocol Specification, Version 2.1 Revision A, Stuttgart: FlexRay Consortium GbR.

Francez, N. (1992) Program Verification, Wokingham, Engleand: Addison-Wesley Publishing Company.

The MathWorks, Inc. (2008) Online Support [online], available: <http://www.mathworks.com/support/> [accessed 8 December 2008].

Vector Informatik GmbH (2007) CANalyzer 7.0, Stuttgart, Germany.

Section IV:
Model Calibration &
Validation

Chapter 13 . Calibration

13.1 Introduction

The calibration process outlined in this chapter focuses on adjusting the model to accurately reflect a real FlexRay node. The chapter describes the process of obtaining timing data from a real world FlexRay node.

All calibration is an iterative process to finely adjust the system. This can be seen in Figure 13.1 (Banks et. al. 2001, p375).

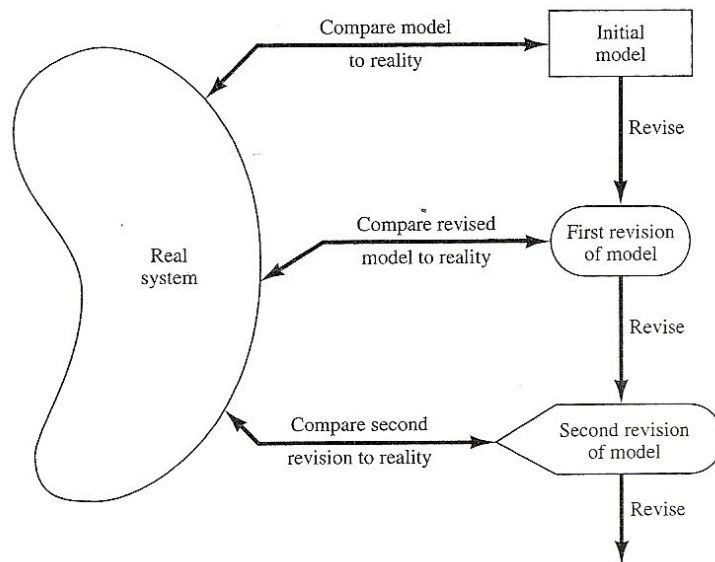


Figure 13.1: Calibration iterative process

The data obtained using the techniques discussed below was used in the validation stage also. Figure 13.2 (Banks et. al. 2001, p16) highlights the stage at which calibration is done in the model development flowchart. In Figure 13.2 the validation stage is highlighted. This is due to calibration being the first stage of the overall validation process. However the process is discussed as a separate step for the purpose of this research.

MODEL CALIBRATION & VALIDATION

This chapter will first describe the equipment that was used to carry out the calibration. Next, the process of obtaining the timing data will be described. The test cases will then be described and the results will be presented. Finally the procedure used will be reviewed.

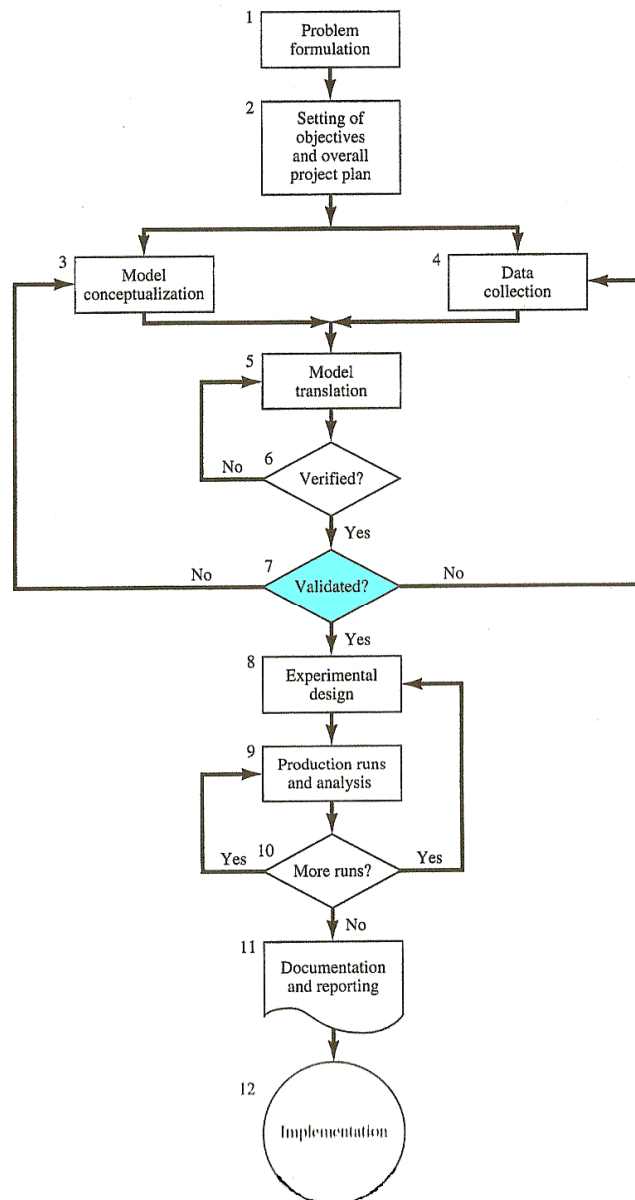


Figure 13.2: Simulation model development process

13.2 Test Equipment

13.2.1 Hardware

This section describes the hardware components to calibrate the simulation model.

13.2.1.1 Fujitsu SK-91F467-FLEXRAY

The Fujitsu SK-91F467-FLEXRAY is a development board. It is designed for development of both the MB88121 FlexRay communication controller and the MB91F467DA 32-BIT Flash microcontroller, both by Fujitsu (Fujitsu Microelectronics Europe 2007b, p7).

The features of the board include (Fujitsu Microelectronics Europe 2007b, p8):

- 5V, 3.3V, 2.5V and 1.8V on-board switching regulators using a 9-12V unregulated DC power supply
- 23Mbit SRAM on-board memory
- In-Circuit serial flash programming
- Three LIN/RS-232 UART interfaces
- Three high-speed CAN interfaces
- Two FlexRay Channel interfaces (Channel A & B)
- Possibility of using FlexRay physical layer driver modules from TZ Mikroelektronik (TZM)
- Status indicators (LEDs) with the option of connecting an alpha-numeric LCD.

The operating speeds of the MCU are up to 100MHz using an external 4MHz crystal oscillator and a PLL built into the MCU. The FlexRay communication controller operates at 10MHz using a crystal mounted in a socket (Fujitsu Microelectronics Europe 2007b, p10). The FlexRay physical connectors, CAN connectors and LIN/RS232 connectors are all 9-Pin D-sub connectors. If the TZM driver modules are not used then the FlexRay communications use RS485 transceivers. Figure 13.3 shows a top down view of the development board (Fujitsu Microelectronics Europe 2007b, p13).

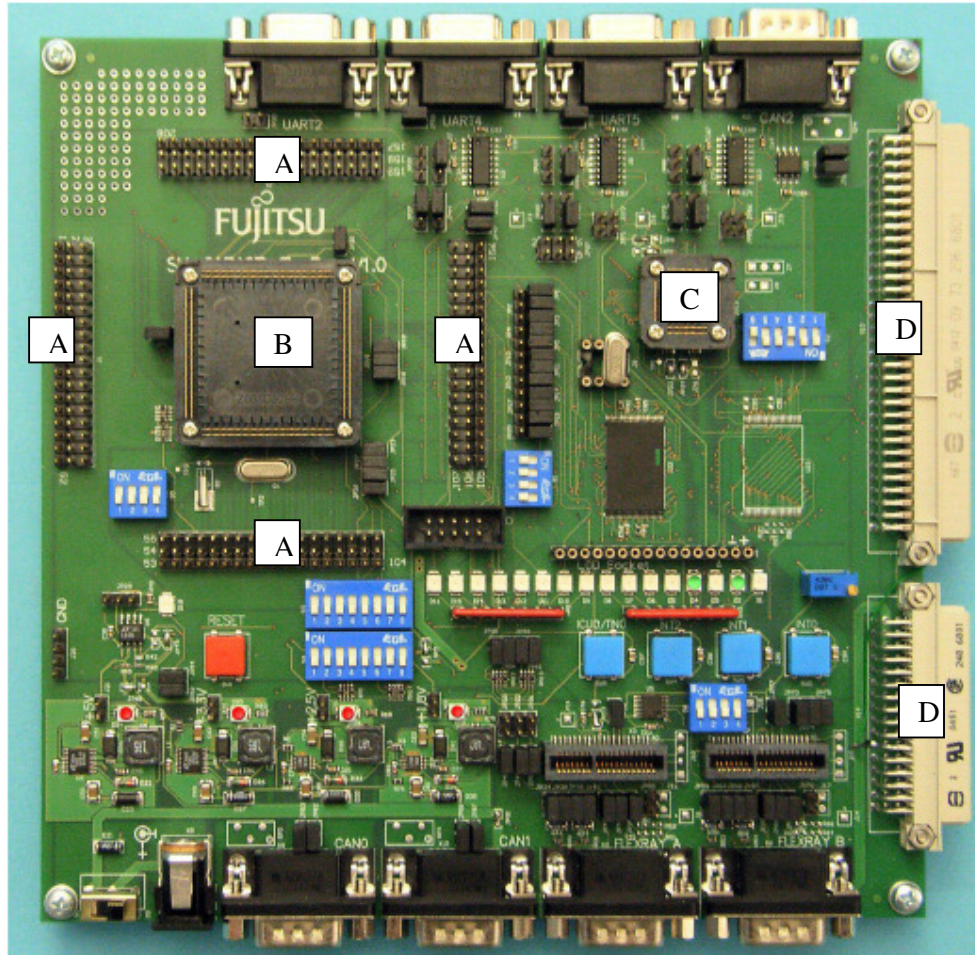


Figure 13.3: Top down view of the Fujitsu SK-91F467-FLEXRAY development board

Figure 13.3 shows edge connectors that are connected to the MCU pins marked with an 'A', the MCU is marked with a 'B', the communications controller is marked with a 'C' and external bus interfaces are marked with a 'D'. The external bus interfaces (D) allow expansions such as Fujitsu graphic device sub boards or user application devices to be connected to the bus of the board (Fujitsu Microelectronics Europe 2007b, p10). The expansion slots for the TZM physical bus driver modules can be seen just above the two bottom right 9-pin D-sub connectors in Figure 13.2. The board also comes supplied with an AC-DC power adaptor with various plug adaptors to suit many type of plug connectors.

13.2.1.2 TZM FlexTiny

The TZM FlexTiny family is a group of interface modules of physical layer drivers for different bus systems (TZ Mikroelektronik 2007b, p7). Some of the different

bus systems that are supported by the FlexTiny family are (TZ Mikroelektronik 2007, p1):

- FlexRay
- CAN
- LIN

They offer different bus termination and/or shielding options. Figure 13.4 (TZ Mikroelektronik 2007b, p1) shows a picture of one of the modules.

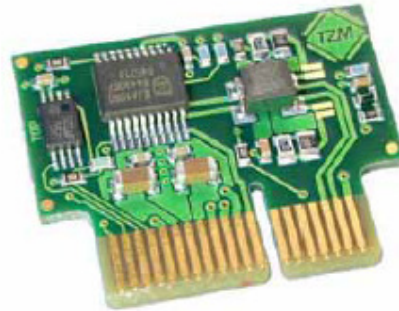


Figure 13.4: FlexTiny module

When these are placed in the development board it is essential that jumpers on the board are reconfigured to ensure that the RS485 transceiver is isolated from the FlexTiny module.

13.2.1.3 TZM Passive Star

To create a FlexRay network two passive star adaptors are available to connect each node of each channel together. Each star has six 9-pin D-sub connectors. The use of a passive star saves money as an active star is not necessary for the tests carried out for this project. A passive star simply takes data sent out on from one node and repeats it on all other lines connected to it. An active star will regenerate the signal to keep the signal strong and is unnecessary for small networks. A picture of the TZM passive star is shown in Figure 13.5 (TZ Mikroelektronik 2004b).

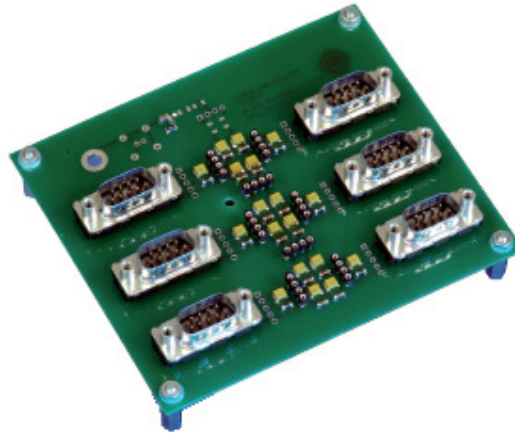


Figure 13.5: Passive star

13.2.1.4 Vector Hardware Interfaces for FlexRay and CAN

The Vector CANalyzer software that is discussed in section 13.2.2, monitors data on a FlexRay network using a hardware interface. This is an interface to the FlexRay physical bus and this is achieved through a Vector VN3600 FlexRay interface module. This connects a computer to a FlexRay network through a USB 2.0 connection. It contains an Intel PXA270 microcontroller operating at 312MHz with 8MBytes of RAM. There are two communication controllers. For startup there is a Fujitsu MB88121B and for analysis a Bosch E-Ray implemented on an Altera Cyclone II EP2C70 (Vector Informatik GmbH 2007d, p5). Figure 13.6 (Vector Informatik GmbH 2007d, p5) shows a VN3600 interface.



Figure 13.6: Vector VN3600 USB interface for FlexRay

13.2.2 Software

This section describes the software used in the project. The software is used to create a simulation as well as to support the hardware.

13.2.2.1 DECOMSYS Designer Pro^{††}

Designer Pro is a design tool that supports the configuration of various FlexRay communication layers. These include (Dependable Computer Systems 2007, p1):

- The frame-based communication layer DECOMSYS::COMMSTACK
- The signal-based communication layer DECOMSYS::FLEXCOM
- The operating system Application Execution System (AES)

There is also the ability to upgrade designs generated using DECOMSYS::DESIGNER using an XCEDF importer.

Designer Pro separates the workflow of the network into the system workflow and the supplier workflow, helping to design the system step by step. There are also controller support modules that allow the design tool to get all information needed for supported controllers. SIMTOOLS is integrated into Designer Pro and adds Simulink blocks for FlexRay design. This helps create a model-based design flow. (Dependable Computer Systems 2007, p1).

13.2.2.1.1 System Workflow

The system workflow allows the system designer to (Dependable Computer Systems 2007, p2):

- Define an architecture specification.
- Define a system specification.
- Define a communication schedule
- Export network design data in FIBEX or .bor data exchange formats.

^{††} In June 2007 Elektrobit Corporation bought Decomsys Beteiligungs GmbH. Since then they have been selling products under the EB Tresos brand. The EB Tresos Designer offers the same interface and abilities as the Decomsys Designer tool.

This system workflow allows all relevant communications cycle parameters to be set. This is then used to set up any node configuration. The system workflow therefore steps the user through the relevant steps to configure the networks parameters. This data can then be exchanged between a number of different system developers.

13.2.2.1.2 ECU Workflow

The ECU workflow allows the system designer to (Dependable Computer Systems 2007, p2):

- Import network data.
- Define a hardware specification.
- Define an application/task assignment to ECU's and application/task definition.
- Configure the driver.

The ECU workflow allows a developer to access information about aspects of the communication cycle such as the assigned slots for a given node. The developer can then configure the node and produce node specific driver configuration files.

13.2.2.1.3 User Interface

The user interface is shown below in Figure 13.7 (Dependable Computer Systems 2007, p3) and consists of three windows:

- An operations window
- A display window
- A log window

MODEL CALIBRATION & VALIDATION

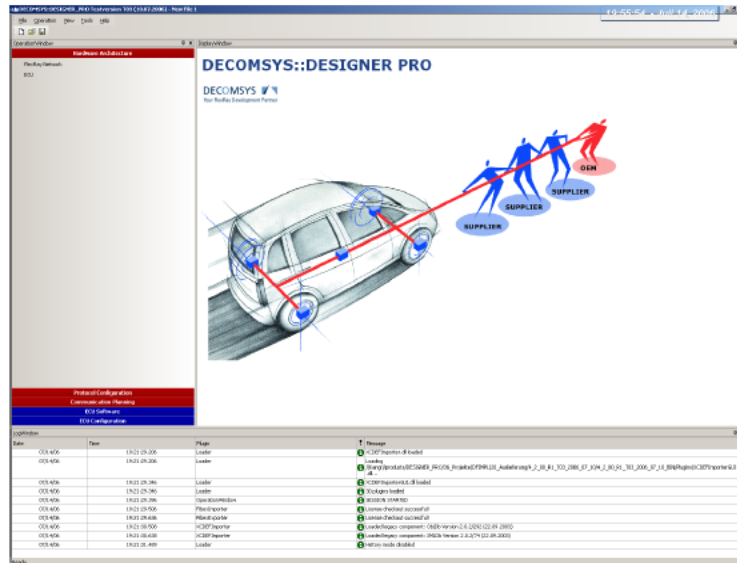


Figure 13.7: Designer Pro main window

The operations window allows selection of either system workflow (shown in red) or ECU workflow (shown in blue) (Dependable Computer Systems 2007, p3).

When setting up various aspects of the FlexRay network there are various constraints that must be met. There is no point in trying to set up a network that has too many or few static slots or in defining a long cycle time unnecessarily. Designer Pro helps the designer to avoid mistakes during design time. This can be seen from Figure 13.8 (Dependable Computer Systems 2007, p13) and Figure 13.9 (Dependable Computer Systems 2007, p12). Figure 13.8 shows the first page of the FlexRay configuration wizard and Figure 13.9 is the second page of the FlexRay configuration wizard. As can be seen in the right-hand column of both screens there is a constraints column. These warn of violations in the configuration of the network. In Figure 13.8 there are a number of constraints flagged (in red) in the right hand column as incorrect. The nodes and frames are also assigned and set up in a similar user-friendly way.

MODEL CALIBRATION & VALIDATION

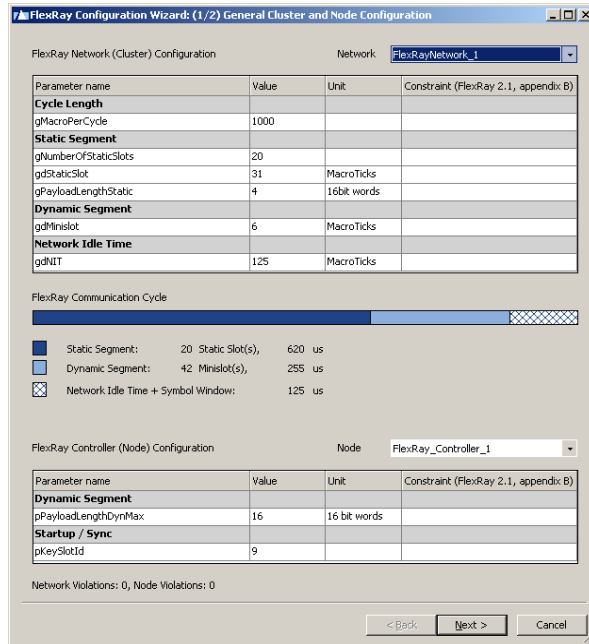


Figure 13.8: The first page of the FlexRay configuration wizard

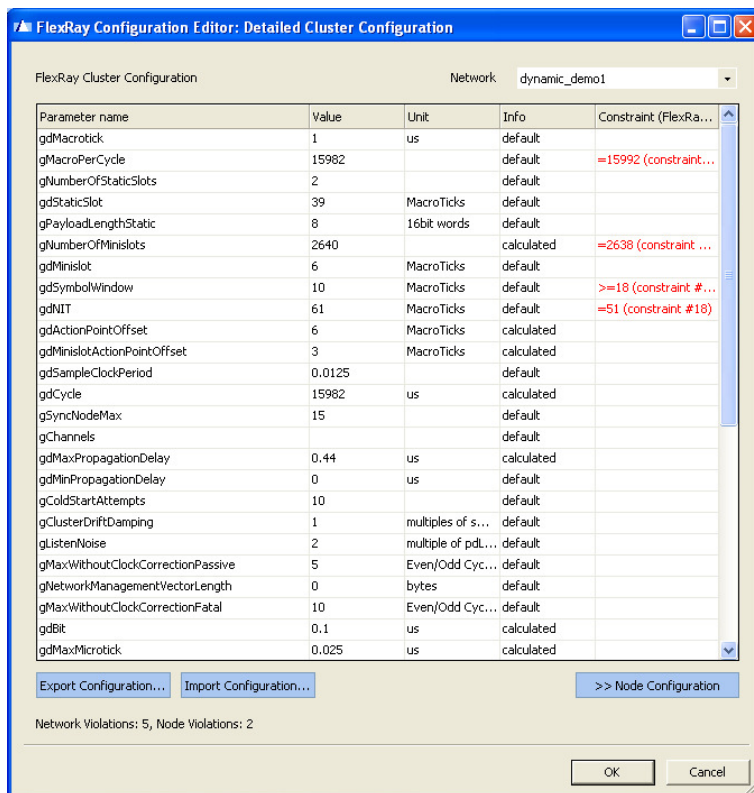


Figure 13.9: The second page of the FlexRay configuration wizard

13.2.2.2 Fujitsu Softune – FR Family Softune Workbench

The Fujitsu Softune Workbench is a tool for developing programs for Fujitsu family microprocessors or microcontrollers. It includes a development manager, simulator debugger, emulator debugger, monitor debugger and an integrated development environment (Fujitsu Limited 2004a, pi). The main window is shown in Figure 12.8. The project visible in Figure 13.10 is an example program from the SK-91F467-FLEXRAY Fujitsu website (Fujitsu Microelectronics Europe 2007c) and is the 91460_static2_91467d-v13 example.

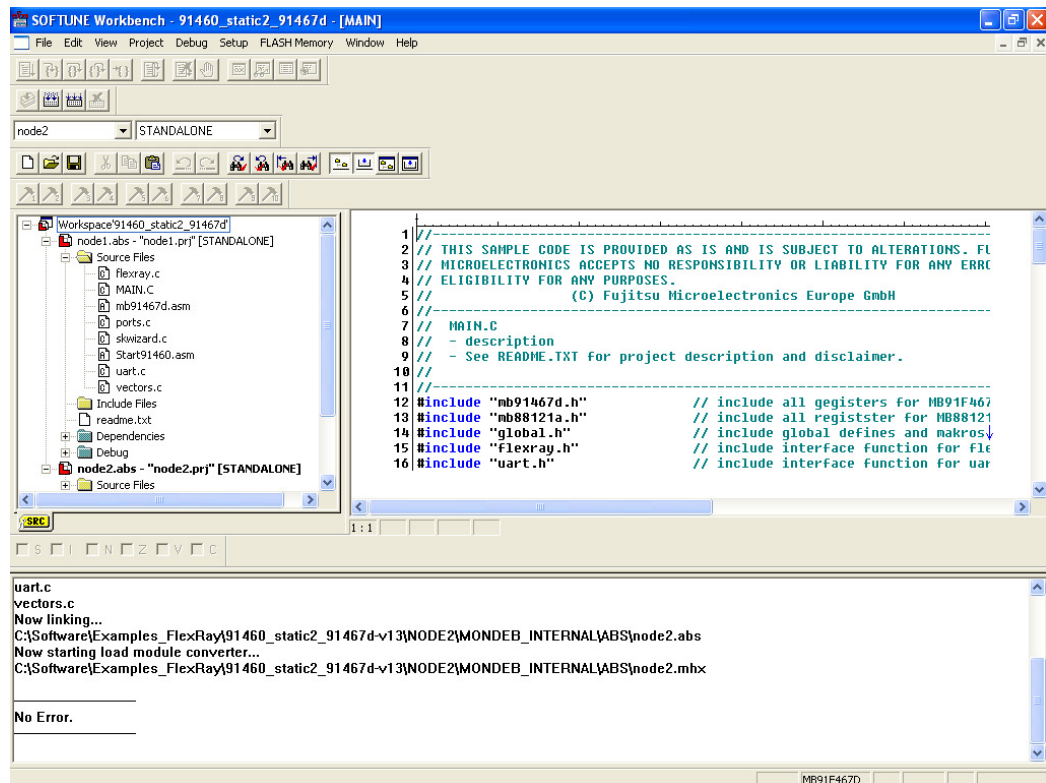


Figure 13.10: Softune Workbench main window

As can be seen from Figure 13.10 the application is broken into three main windows:

- A window to select files to edit (top left)
- An editor window (top right)
- An error reporter below

The file selection window allows you to see what files are currently contained in your project and select a file to edit. The editor window allows editing of the code.

Finally the error report window displays any errors that are detected during a make or build of a project.

13.2.2.3 TZM FlexConfig

The FlexConfig tool is used to configure FlexRay communication controllers. The parameters are set and checked against limits or constraints to eliminate possible configuration errors (Fujitsu Microelectronics Europe 2007a, p58). FlexConfig allows users to generate all the configuration files for each node in a cluster including node specific buffer settings (TZ Mikroelektronik 2007a, p1). There are also different types of communication controller supported, such as the Bosch E-Ray and the Freescale FPGA 9.1. Figure 13.11 shows the FlexConfig main window.

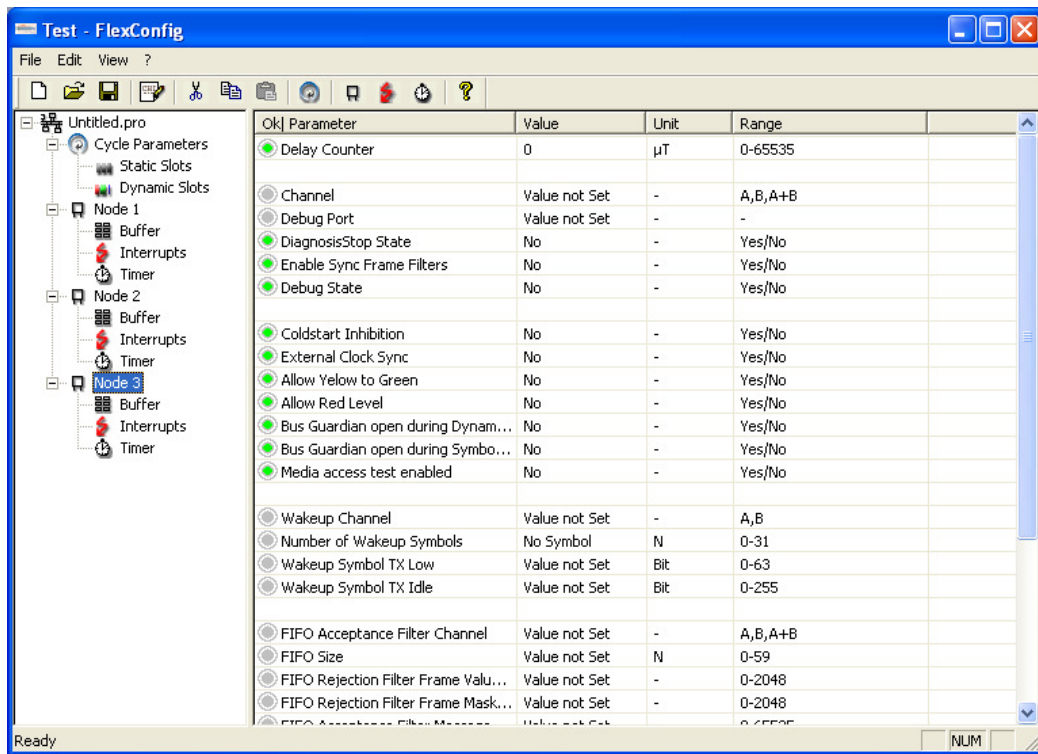


Figure 13.11: FlexConfig main window

13.2.2.4 Vector CANalyzer.FlexRay

CANalyzer is an analysis tool for networks or distributed systems. It supports CAN, LIN, MOST and FlexRay systems, allowing the user to configure the system to

MODEL CALIBRATION & VALIDATION

observe and analyse traffic patterns for specific network implementations and constraints (Vector Informatik GmbH 2007b, p1).

The basic functions provided by CANalyzer include (Vector Informatik GmbH 2007b, p1):

- Listing of bus traffic.
- Graphical and textual representation of network traffic values.
- Sending predefined messages.
- Sending logged messages.
- Statistics on messages.
- Bus Load and disturbance statistics.
- Generation of bus disturbances.

The data observed can also be logged in files allowing for retrieval of experiment data for offline analysis. The logged information can also be confined to specific time windows or given event triggers (Vector Informatik GmbH 2007b, p3).

CANalyzer.FlexRay needs to be configured to access a FlexRay node. To do this CANalyzer.FlexRay allows several methods for importing configuration data such as manual input of the network data or importing .chi files. It is also compatible with several PCI interfaces as well as the VN3600 USB interface. This allows easy integration into a FlexRay network under study (Vector Informatik GmbH 2007a, p2). Figure 13.12 (Vector Informatik GmbH 2007b, p1) shows the CANalyzer.FlexRay trace window.

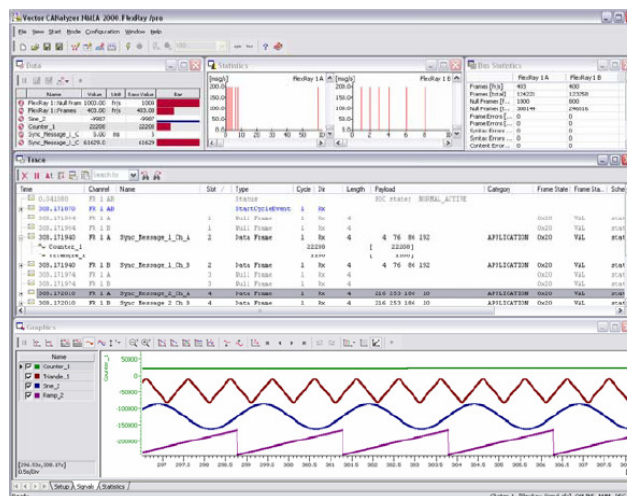


Figure 13.12: CANalyzer.FlexRay trace window

13.2.2.5 MATLAB, Simulink and SimEvents

The simulation model will be built and run using MATLAB, Simulink and SimEvents. MATLAB, Simulink and SimEvents are covered in detail in section 7.9 of this thesis.

13.3 Calibration Procedure

The first step in any calibration procedure is to obtain the timing information to calibrate the model. When the timing data is obtained the model can be tested to see if it can react in the same way as the real world system. Modifications to the model can then be made to more accurately reflect the real world system.

13.3.1 Gathering Timing Information

To obtain the timing parameters of an E-Ray chip the following techniques were used. These techniques were applied to the remainder of the components as outlined in sections 13.3.2 to 13.3.7 (inclusive). The technique was necessary as CANalyzer is unable to give timing information for the operation of the E-Ray chip. To gain timing data for these operations it was necessary to time them with the use of a free run timer and interrupts. A sample program from Fujitsu was obtained and modified for each calibration test case. The sample programs used were called '91460_dynamic1_91467d-v16' (Fujitsu Microelectronics Europe GmbH 2007e) and '91460_dynamic_int1_91467d-v15' (Fujitsu Microelectronics Europe GmbH 2007f). These programs are set up to both use the FlexRay Driver (FFRD) and the '91460_dynamic1_91467d-v16' program was set up for use with the DECOMSYS::COMMSTACK<FlexRay> v1.8 software driver also.

The samples both use a reload timer to synchronise a time-triggered task to the FlexRay cycle. The value of the reload timer is checked against the current communication cycle time at regular intervals. This is to ensure there is no drift between the two values as they do not share a common clock. When a push button is pressed a counter value is updated and this data is transmitted over the static segment of the communication cycle. Another button is connected to the analogue to digital converter (ADC). The data obtained from the ADC is transmitted across the physical bus during the dynamic segment.

For the messages assigned to receive buffers the communications controller automatically stores the information. The host must then check this data. The '91460_dynamic1_91467d-v16' program does this automatically every communication cycle. The '91460_dynamic_int1_91467d-v15' program reacts to an interrupt generated when the assigned dynamic message is received. The program will then check the received message buffer.

Both programs were needed to calibrate the software model. The '91460_dynamic_int1_91467d-v15' was used as it was designed to work with the FFRD and was initialised already for use with interrupts. The '91460_dynamic1_91467d-v16' was also used as it could be used with the COMMSTACK software driver. The two could then be compared and reasonable timing data could be obtained for both the software driver and the E-Ray communications controller. The '91460_dynamic1_91467d-v16' could also be used with the FFRD and interrupts were enabled for this purpose. However the '91460_dynamic_int1_91467d-v15' had already been setup at this stage and so was used to separate the two sets of tests. The use of separate programs also helps to reduce confusion when switching between the different tests for the different software drivers.

13.3.1.1 E-Ray Interrupts

The structure of the E-Ray communications controller is shown in Figure 13.13 (Robert Bosch GmbH 2006, p14). The information needed to calibrate the model concerned the time it took information to be generated by the application and sent, through the software driver and communications controller, to the physical bus. As the information passes between the communication controller and host controller, and the communications controller and the physical bus, it is handled by a number of buffers. As FlexRay uses a TDMA scheme any message must be stored before its allocated time slot comes around. To do this there is a message RAM as well as a variety of input and output buffers.

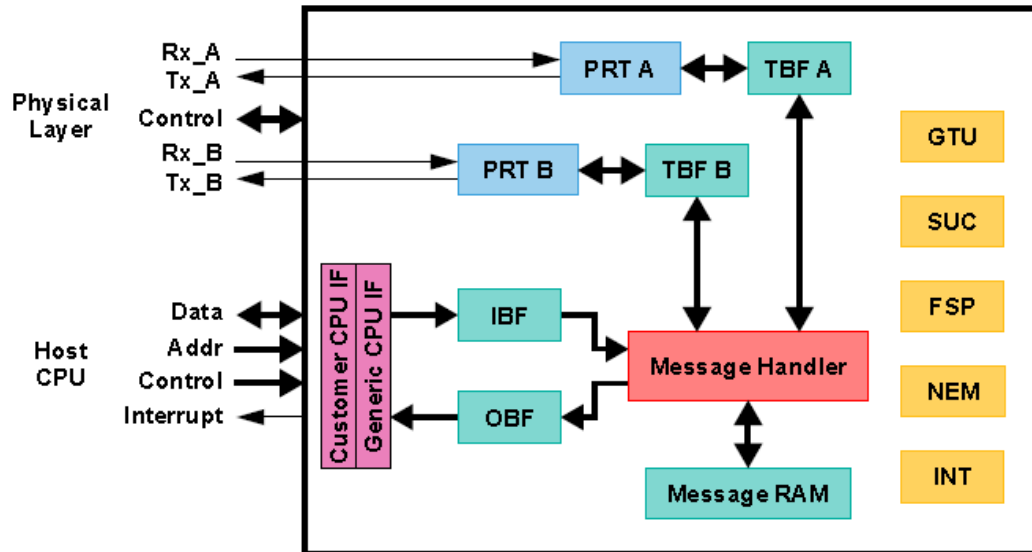


Figure 13.13: E-Ray structure

The time it takes a message to pass through each stage will vary. This time will be affected by the size of data to move as well as the implementation of a particular transfer stage. To be able to determine the time it takes for each stage it was necessary to time each stage. To do this interrupts were used.

The E-Ray is designed to provide information in the form of interrupts. These interrupts can be used by a developer to create a dynamic program that reacts to different circumstances. They can also be used to analyse the performance of the system. An interrupt can be set to generate when one of the following happens (Robert Bosch GmbH 2006, p 147):

- An error occurs.
- A status change is detected.
- A timer is asserted.
- A stop watch event occurs.

The interrupts that can be used to check the timing performance are the status interrupts. There are 20 status interrupts but not all the interrupts usable for the timing tests. Figure 13.14 shows the E-Ray status interrupt register (SIR) (Robert Bosch GmbH 2006, p 27).

MODEL CALIBRATION & VALIDATION

Bit		31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
SIR	R	0	0	0	0	0	0	MTSB	WUPB	0	0	0	0	0	0	MTSA	WUPA
0x0024	W																
Reset		0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Bit		15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	R	SDS	MBSI	SUCS	SWE	TOBC	TIBC	TI1	TI0	NMVC	RFCL	RFNE	RXI	TXI	CYCS	CAS	WST
W	W																
Reset		0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 13.14: E-Ray status register interrupts

The SIR register is covered in chapter 6. The interrupts of interest in the SIR will be briefly covered here. The interrupts of interest are (Robert Bosch GmbH 2006, pp. 27-8):

- Transmit Interrupt (TXI): This interrupt occurs when a successful transmission of a frame occurs.
- Receive Interrupt (RXI): This interrupt occurs when a New Data flag is set for a receive buffer.
- Transfer Input Buffer Completed (TIBC): This interrupt occurs whenever a transfer between the Input buffer and Message RAM is completed.
- Transfer Output Buffer Completed (TOBC): This interrupt occurs whenever a transfer between the Message RAM and Output buffer is completed.

To ensure that multiple interrupts do not occur unexpectedly for interrupts TXI and RXI it is also necessary to set the message buffer interrupt flag (MBI) bit in a message buffers header. This ensures that only a message buffer with MBI set will generate an interrupt. This makes analysis easier to conduct and ensures that an interrupt for a particular message can be tracked.

13.3.1.2 Interrupt Information

The Fujitsu SK-91F467-FLEXRAY has a number of external interrupt lines connecting the MB91F467D host controller to the MB88121 communications controller (the E-Ray chip). According to the ‘readme’ file (Fujitsu Microelectronics Europe GmbH 2007f), supplied with the 91460_dynamic_int1_91467d-v15 program, the interrupt line connections between the host and communication controller chips are as shown in Table 13.1.

Host (MB91F467DA)	E-Ray (MB88121A)
ext. Int4	Int0
ext. Int5	Int1
ext. Int6	Int2

Table 13.1: SK-91F467-FlexRay development board interrupt connections

The readme file also states that interrupt signal of the MB88121A is active high. It should also be noted that the status interrupts are usually assigned to interrupt line 1 while error interrupts are assigned to line 0 (Fujitsu Microcontroller Info Team 2008).

When all this information is gathered it is then possible to setup the MB91F467D host controller to accommodate this. As status interrupts are being used, the MB91F467DA external interrupt 5 should be initialised using rising-edge detection, the port assigned to an external interrupt input and external interrupt 5 enabled. The interrupt vector table external interrupt 4 and 5 priority should be assigned to a value between 16 and 30 (inclusive) to enable the interrupt (Fujitsu Limited 2004b, p313).

When the interrupt is detected the `ffrd_api_interrupt_line1()` function should be assigned to handle the interrupt. This is a function provided by the FFRD and clears the interrupt line flags. It also checks which interrupt was asserted and calls the appropriate handler (Fujitsu Microelectronics Europe (2007d).

The FFRD is used instead of the COMMSTACK interrupts for these levels of interrupts. This is because the FFRD provides interrupt services to handle the desired interrupts. The COMMSTACK only provides interrupts associated with the absolute timer, relative time and cycle start interrupts (Dependable Computer Systems 2006, p15).

13.3.1.3 Timers Used

In order for calibration data to be extracted from the programs a way to measure the time was needed. The FFRD provides a `'get_time()'` function. This function returns the approximate global time in microseconds. This time stamp can be used when a message is sent or received. This time would then reflect the time it takes for a message to pass into/from the physical bus to the chip. The offset from the expected start time of the time slot would then be used to calculate the exact value. This function will take a fixed time to calculate the current global time. To calculate the time for messages to

pass into the host MCU would require two time stamps to achieve similar calculations. To overcome this problem another method was proposed.

The Fujitsu MB91460 Series microcontrollers have a number of features such as an analogue to digital converter (ADC), a digital to Analogue converter (DAC) and reload timers. It also provides a number of 16-bit free-run timers. These free-run timers can be setup in various ways and controlled by software. A free-run timer was therefore set up to count up and the value was then read after an operation was completed. These were operations such as transferring data between the host and the communications controller which could then be accurately timed. Figure 13.15 shows the various registers and possible settings for the free-run timer (Fujitsu Limited 2004b, p736).

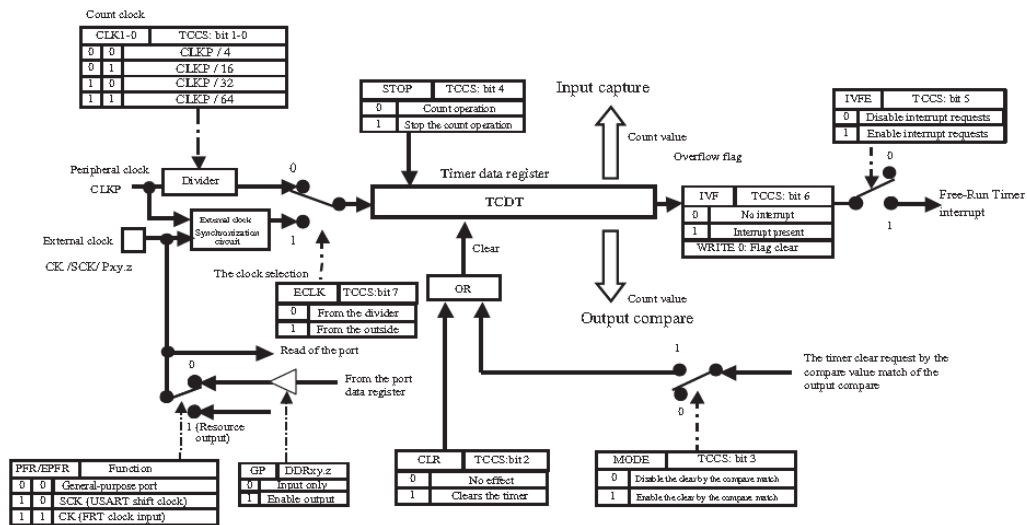


Figure 13.15: Free-run timer settings

Before the free-run value could be used it was necessary to know what each count tick represented. It was known that the host clock was operating at 16Mhz. The free-run timer was set to count every 4 clock ticks. At 16Mhz the clock period is 62.5ns. This gives a time of 250ns for every count of the free-run timer.

13.3.1.4 Hardware Configuration

The hardware that was used was a single SK-91F467-FlexRay development board, two passive stars and a Vector VN3600 FlexRay interface connected to both the passive stars and a laptop. The laptop was used to run the CANalyzer.FlexRay software

MODEL CALIBRATION & VALIDATION

that controls the VN3600 interface. The SK-91F467-FlexRay development board was fitted with two FlexRay versions of the FlexTiny physical layer drivers (one for each FlexRay channel). A second computer was also used to load the application software onto the development board. This was used to load the board with the test programs and to monitor the system. The hardware that was used was set up as in Figure 13.16.

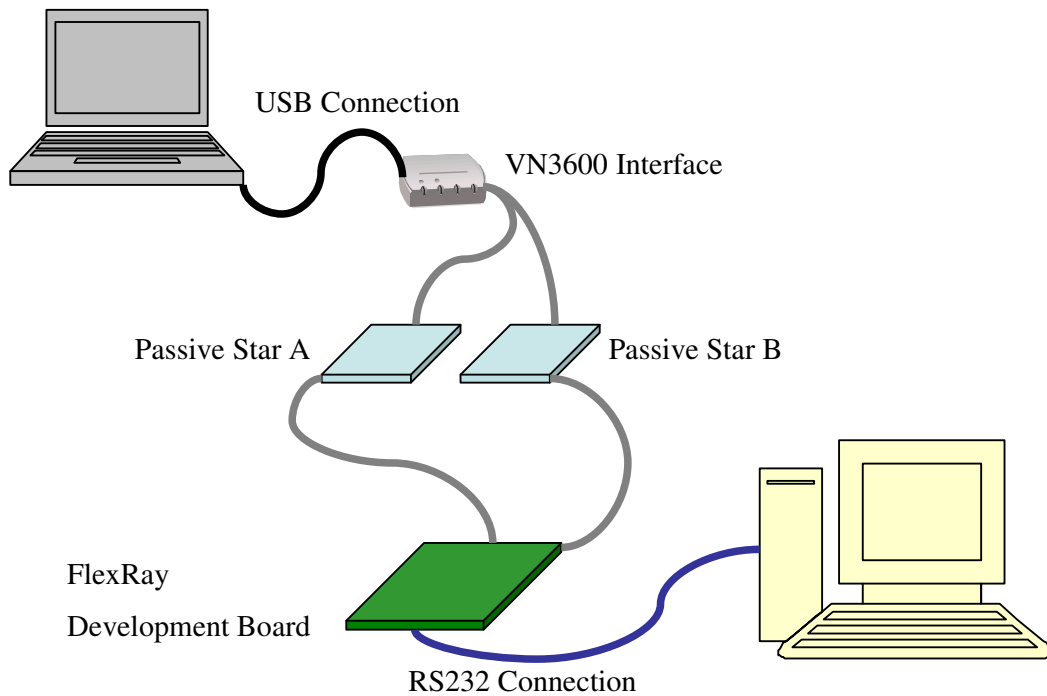


Figure 13.16: Calibration hardware setup

The passive stars were used to allow for additional nodes to be connected if desired. It is not necessary to use them when only connecting two nodes together.

13.3.1.5 MONDEBUG

MONDEBUG is the monitor debugger mode of Softune Workbench. To perform the monitor debugger debugging, the target monitor program must be placed into the target system. The monitor debugger software then communicates with the host computer using a COM port of the computer. A monitor program must be ported to the target hardware (Fujitsu Limited 2004a, p120). When a valid monitor program is loaded the status LEDs will be as shown in Figure 13.17 (Fujitsu Limited 2004b, p58).

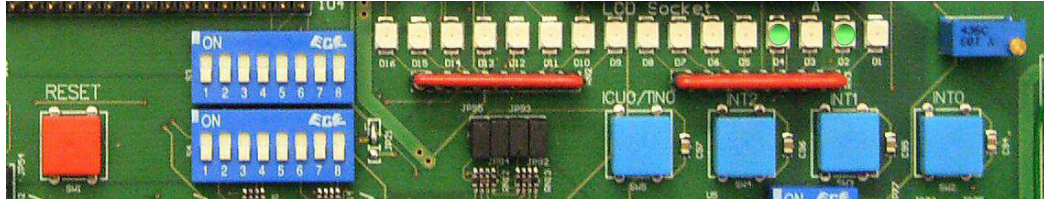


Figure 13.17: Flow directions of data in a FlexRay system

This mode of operation allows the user to use breakpoints and watch windows to monitor the status of various system of the developed program. This allows the developer to avoid using UARTs and COM ports to transfer data which may affect the timing of the system. Variables used to monitor the timing of a system can simply be checked after a breakpoint has been reached.

13.3.1.6 Timing Procedure

A procedure to measure the time taken to transfer data in the chip was developed in the following way. Data transferred to the chip was split into two directions. The first direction was called 'up' and the second was called 'down'. The up direction referred to the data that is obtained from the physical bus and passed to the host via the communications controller. The down direction was the flow of data from the host computer to the communications controller and finally to the physical bus. This is illustrated in Figure 13.18. Host A is receiving data from the physical bus so data is flowing in the up direction. Host B is transmitting data to the physical bus and so the flow of data is in the down direction.

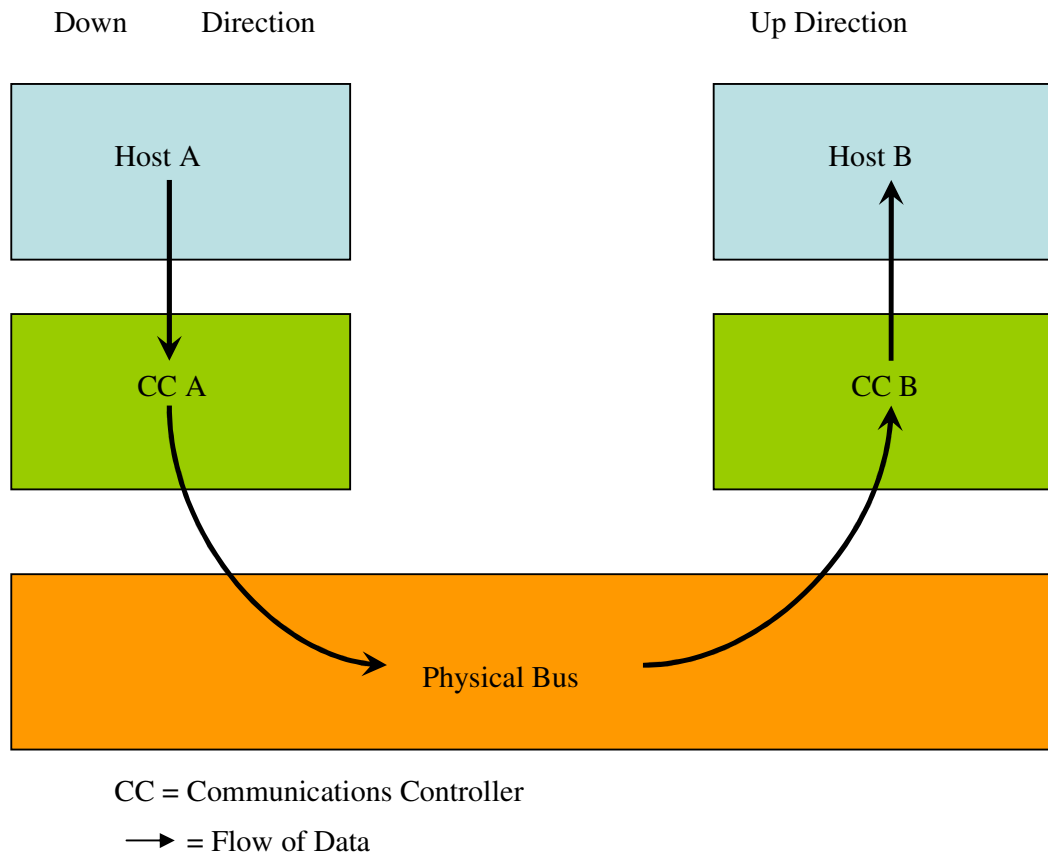


Figure 13.18: Flow directions of data in a FlexRay system

When this distinction was made a clearer work flow was developed. It was also clear from the setup procedure of the interrupts that this distinction would be an advantage. As the MBI bit of a message buffer must be set then a single message could be tracked. This meant that different payloads could easily be tested by switching this bit on for different messages with different payloads.

13.3.1.5.1 Data Flow Timing Tests (Up Direction)

This concerns the receive side of the communication process. Therefore one message buffer should have its MBI bit set to logic 1. When the interrupt occurs indicating that a new message has been received, a time stamp of the current communication global time can be taken. As the node will have transmitted during a given slot, the start of the slot time will be known. The time to receive a frame can be calculated. Also if the payload size is known then a metric can be derived based on this value i.e. the time per byte.

MODEL CALIBRATION & VALIDATION

The next stage would be to get the node to request the data stored in the message buffer. Just before this is done the free-run timer should be started. When the Output buffer has received data from the message handler a time stamp can be recorded from the interrupt using the free-run timer. Next when the software driver has signalled it has successfully received the data, another time stamp could be obtained.

This can be repeated for a number of different payload sizes along with different FlexRay communication configurations. This ensures that accurate measurements can be made. The different FlexRay communication configurations will allow the testing of any possible effect that a different configuration may produce.

The time it takes the COMMSTACK driver to return from a read buffer command should also be made. This can be compared to the FFRD timings and a reasonable execution time can then be obtained for the COMMSTACK.

13.3.1.5.2 Data Flow Timing Tests (Down Direction)

The transmit aspect is handled by the node by passing information flowing down through the node. Again one message buffer should have its MBI bit set to '1'. The first stage would be to get the host to update a transmit message buffer. Firstly though, the free-run timer should be started just before the host attempts to update the buffer. When the Input buffer of the communications controller interrupts the program the free-run timer can be checked and timing data can be obtained for this operation.

When the buffer indicates that an update of its contents has occurred then another time stamp can be obtained from the free-run timer. When the software driver indicates a successful transfer of data to the communications controller, then a final time stamp should be obtained from the free-run timer.

Finally the transmit interrupt should be setup. When this interrupt occurs then the current global communication time should be recorded. This can then be compared to the start time at which the slot should begin.

This can be repeated for a number of different payload sizes along with different FlexRay communication configurations. This ensures that accurate measurements can be made. The different FlexRay communication configurations will allow the testing of any possible effect that a different configuration may produce.

The time of it takes the COMMSTACK driver to return from an update buffer command should also be made. This can be compared to the FFRD timings and a reasonable execution time can then be obtained for the COMMSTACK.

13.4 Calibration Test Cases

In order to be useful the different test case configuration parameters needed to be converted into setup files for the hardware. In the case of CANalyzer and the FFRD a '.chi' file was needed. The COMMSTACK software driver uses DECOMSYS Designer files to set up the communications controller. To create these Designer and FlexConfig were used. However these two programs configure the settings in different ways.

Tests were run with both configurations and it was discovered that the FlexConfig-generated configurations wouldn't always allow synchronisation with the hardware devices. The Designer configurations worked as expected however. New chi files were then constructed with the buffer assignments from the FlexConfig software and the other constraints from the Designer files. These were tested and worked as desired. The buffering was tested separately also to ensure the right configuration had been achieved.

At this stage the configuration of the Test Case 6 was incompatible with real world configurations. The FIFO rejection filter could not be configured as desired. Test case 6 could therefore not produce the required timing information. Table 13.2 shows the test case parameters for each calibration test. This includes the updated Test 6 FIFO parameters.

MODEL CALIBRATION & VALIDATION

ID	Cycle Length (µs)	Number of Static Slots	Number of Mini Slots	Static Slot Length (µs)	Mini Slot Length (µs)	Static Frame Payload (words)	Dynamic Frame Payload (words max)	Channels	NIT & Symbol Length (µs)	Node Tx Frames	Node Rx Frames	Latest Tx	Note
1	16000	630	0	25	NA	1	NA	A&B	250	3 and 44	6 and 18	0	Maximum cycle length with no mini slots
2	16000	2	1548	43	10	10	20	A&B	431	2 and 444 (A)	1 and 181 (B)	1543	Maximum cycle length, Minimum static slots
3	5000	20	209	123	10	50	80	A&B	447	3 and 65 (A)	6 and 66 (B)	192	Medium cycle length, medium number of static slots, medium number of mini slots
4	5000	17	34	278	6	127	5	A&B	67	3 and 28 (A)	6 and 29 (B)	29	Large static slot size, small number mini slots of mini slots
5	114	2	0	27	NA	2	NA	A&B	60	2	1	0	Minimum cycle length
6	5000	60	239	36	10	6	20	A&B	447	3 and 65	61 (B)	234	All received messages stored in FIFO

Table 13.2: Calibration test case parameters

MODEL CALIBRATION & VALIDATION

Again for the model tests it was necessary to run the model with repeatable results. The seeds that were used can be seen in Table 13.3.

Test Case	Application Generation	Application Response	Physical Bus A-1	Physical Bus A-2	Physical Bus B-1	Physical Bus B-2
1	901	1763	272	777	15973	12
2	231	405706	19881	3103	568	86418
3	6068	93	1527392702	790176266	708	6382325
4	860	916904	4000676564	72711	411327	284444
5	891	270206	432287947	159790	93808	1001
6	7621	8	1578461665	9797	173	63272

Table 13.3: Calibration test case random number seeds

13.5 Calibration Data & Results

13.5.1 Calibration Data

When the collection of data was carried out, it was noted that a single development board would not easily synchronise with the VN3600 correctly. Test Case 2 was the only configuration that worked sufficiently. Vector (the CANalyzer developers) were contacted for help and feedback obtained to try to resolve the issue. A possible reason for this problem was the exact implementation of the two hardware devices (the VN3600 interface and SK-91F467-FLEXRAY development board) were not quite compatible. Another possible answer is that the CANalyzer equipment was not properly configured. It was felt that it would be more beneficial to have at least two development boards running the same configuration and application software. Another SK-91F467-FlexRay development board was obtained and the issue was no longer present in the setup. The revised hardware configuration can be seen in Figure 13.19.

MODEL CALIBRATION & VALIDATION

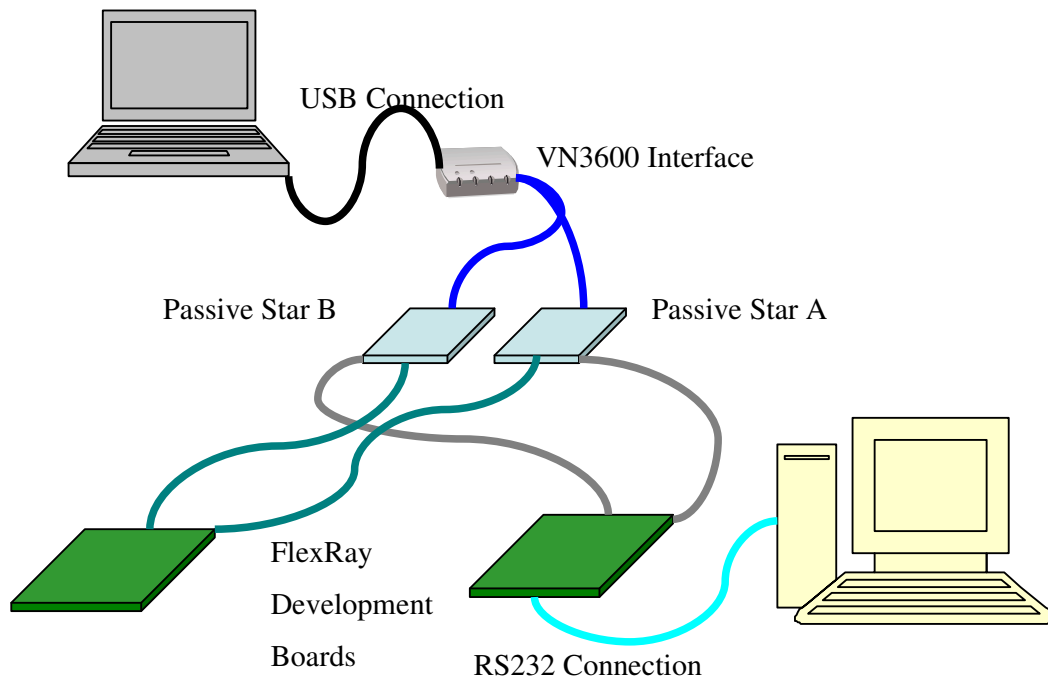


Figure 13.19: Calibration hardware setup – revised

A single serial RS232 connection was used to program the development boards with appropriate application software. This allowed for greater control of when information was sent or received from the boards. The data obtained is outlined below. For each measurement and payload length ten timings were taken.

The timing data that was desired to be obtained for the model was as follows:

- The software driver transmission time
- The software driver receive time
- The time for the IBF to pass data to the Message RAM
- The time for the OBF to receive data from the Message RAM
- The time to transfer data to a transient buffer from the Message RAM
- The time to transfer data to the Message RAM from a transient buffer
- The Message RAM buffer update time
- The Message RAM read time

13.5.1.1 Interrupt latency and Time Function Timings

The timing data gathered for the calibration tests was taken from real world observed times. These were obtained from a SK-91F467-FLEXRAY development board. The time for the communications controller to signal a specific condition (i.e. an

MODEL CALIBRATION & VALIDATION

interrupt) will take a given time. Any program that is currently being executing on the host controller will also need to be halted before the interrupt handler can be executed. To ensure that the program can continue correct execution after the interrupt is serviced a context for the program running must be saved. This all takes time and is known as the interrupt latency. It is important to know how long this takes to ensure accurate timing information can be obtained. To get accurate timing information it is also necessary to know how long a function takes to get a time stamp. This can then be accounted for when analysing the timing information obtained from the development board.

Two timing functions (the `ffrd_api_get_time()` function and free run timer functions) were used to obtain timing information from the board as was stated in 13.3.1.3. To obtain accurate execution times for the timers, ten timestamps were taken one after another during the program flow. Each time value was stored to a different element of an array. To ensure that an accurate time was obtained, each function was written explicitly. This ensured that no delay would occur due to the implementation of a ‘for’ loop, for example. An example of this is as shown as follows:

```
Time[0] = ffrd_api_get_time();  
Time[1] = ffrd_api_get_time();  
.....  
.....  
Time[9] = ffrd_api_get_time();
```

The times were then compared to each other and the difference in time calculated. The difference in time between two consecutive time stamps is the time taken to execute the function. This was done for both timers for each of the test cases. This ensured that a large enough number of runs were done to allow for any false readings. These could be from an interrupt happening between two of the functions being executed. This would cause a larger time than usual. Tables 13.4 and 13.5 show the execution times of the FFRD `ffrd_api_get_time()` function and the free-run timer functions.

MODEL CALIBRATION & VALIDATION

Test Case	1	2	3	4	5
	67	66	66	66	66
	66	67	65	66	67
	66	66	66	66	66
	67	66	66	65	63
	66	66	66	66	69
	66	67	66	66	67
	66	66	66	66	66
	67	66	66	66	66
	66	67	66	66	67

Table 13.4: ffrd_api_get_time() time differences (μs)

Test Case	1	2	3	4	5
	1500	1500	1500	1750	1500
	1750	1750	1750	1500	1750
	1500	1500	1500	1750	1500
	1750	1750	1750	1500	1750
	1500	1500	1500	1750	1500
	1750	1750	1750	1500	1750
	1500	1500	1500	1750	1500
	1750	1750	1750	1500	1750
	1500	1500	1500	1750	1500

Table 13.5: Free run timer differences (ns)

Based on these obtained times it was concluded that the `ffrd_api_get_time()` function takes roughly 66 μs to obtain a time. The free run timer can also obtain a time stamp about every 1500 ns based on the setup of the system. These values are taken from analysing the mean, median and mode of the differences observed.

The time for an interrupt to be detected and the interrupt handler to be run was also tested. To do this the ‘Timer 0’ absolute timer of the E-Ray was used. This timer can be configured to interrupt at a specific macrotick time during all or specific communication cycles. At the specific macrotick that the timer interrupts the host and as the execution time of the `ffrd_api_get_time()` function was known, an accurate interrupt latency could be calculated. When these times were obtained the latency times were calculated by subtracting the interrupt Timer 0 configured macrotick value and the `ffrd_api_get_time()` execution time. The observed interrupt latency times are shown in Table 13.6.

MODEL CALIBRATION & VALIDATION

Test Case	1	2	3	4
	64	64	64	64
	63	64	64	64
	64	64	64	64
	63	64	63	64
	64	64	63	64
	63	64	63	64
	63	64	63	64
	63	64	64	64
	63	64	63	64
	64	64	63	64

Table 13.6: Interrupt latency times (μs)

The value of 64 μs was taken as the interrupt latency based on the values observed. This value was taken into account for all relevant calibration timing data recorded.

13.5.1.2 Fujitsu FlexRay Driver Transmit Times

The time to correctly transfer data between the host and communications controller was measured. This is the time it takes the software driver to pass the information to the communications controller. The diagram below shows a strong relationship between the size of the data to be transferred and the time taken to transfer the data. As the size of data increases the time to pass on the information increases also. Figure 13.20 is a graph of times observed versus the number of data bytes passed from the application layer to the communications controller.

MODEL CALIBRATION & VALIDATION

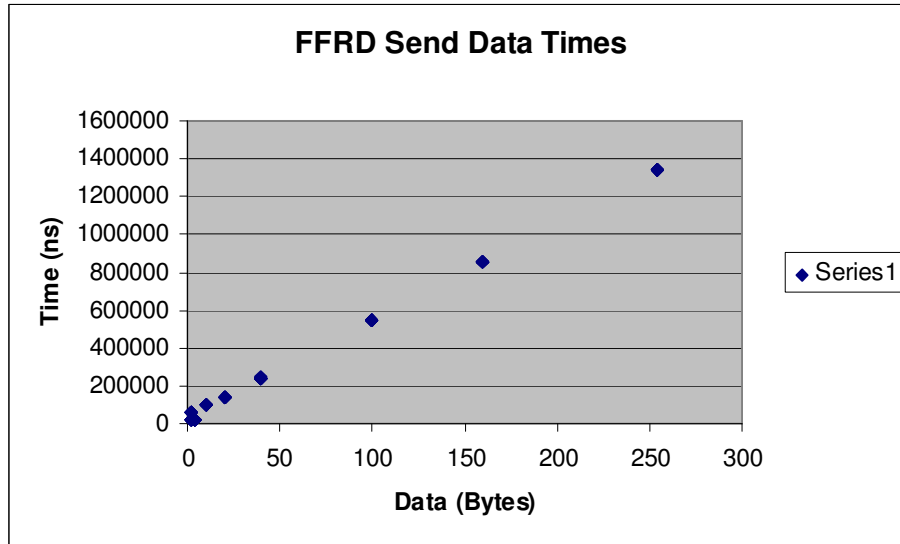


Figure 13.20: Fujitsu FlexRay Driver transmit timings

The trend line function of Microsoft Excel was used to place a trend line on the graph. It also calculates an R^2 value. This value is a relationship between how much the x values, i.e the data size, changes compared to how much the y values changes, i.e. the deviation in the time taken,. An R^2 value of '1' means that there is a direct relationship between the two values and one value effects the other. On the other hand an R^2 value of 0 means that there is not a relationship between sets of values, therefore as one set of values changes the other set remains the same (Freud 1979, p394). A graph of software drivers transmit times including a trend line is shown in Figure 13.21.

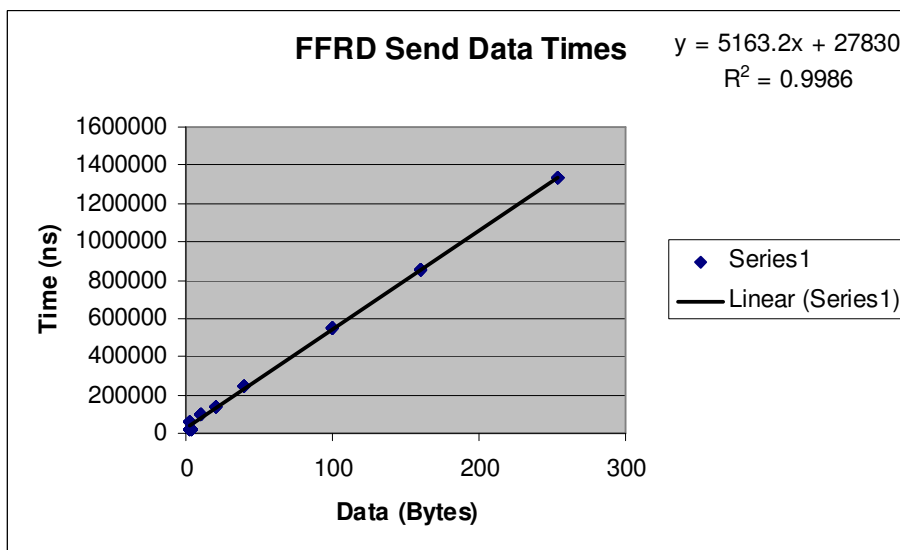


Figure 13.21: Fujitsu FlexRay Driver transmit timings with linear trend line

With the R^2 value being close to 1, this indicates that there is a strong relationship between the payload size and the transfer time. A number of trend line types were tested and the highest value of R^2 was taken as the correct relationship trend. Figure 13.22 shows a polynomial trend line as the second trend line type that had the same R^2 value.

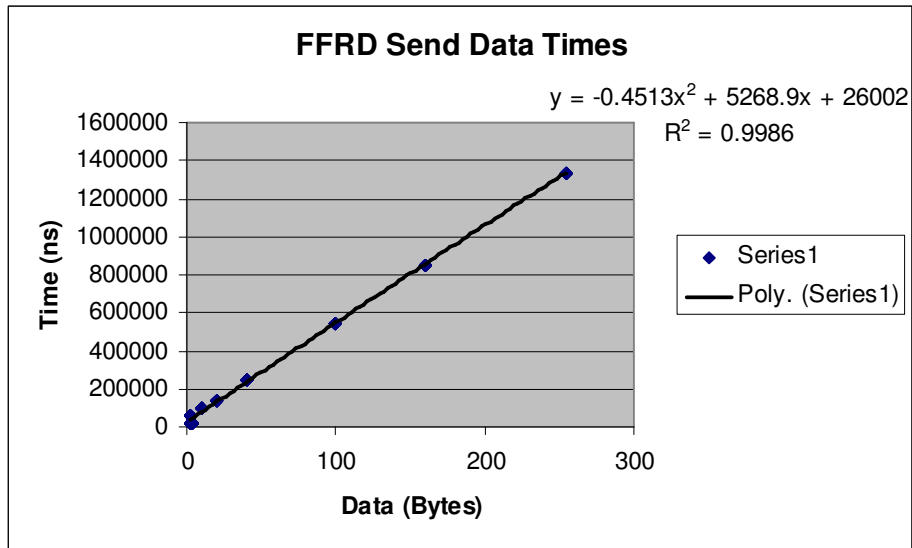


Figure 13.22: Fujitsu FlexRay Driver transmit timings with polynomial trend line

There is no significant difference between the two formule for the trend. Both maintain an offset and a time factor based on the size of data to transfer. The polynomial formula is more computationally intense however and so the linear trend line formula was taken as the correct relationship formula. This formula was therefore used in the simulation model to calculate the delay for the FFRD software driver based on the amount of data bytes to be transferred.

13.5.1.3 Fujitsu FlexRay Driver Receive Times

The graphs of the FFRD receive times show a strong relationship between the size of the data to be transferred and the time taken to transfer the data. In Figure 13.23 the relationship between the time taken to obtain the data from the communications controller and the data size is shown.

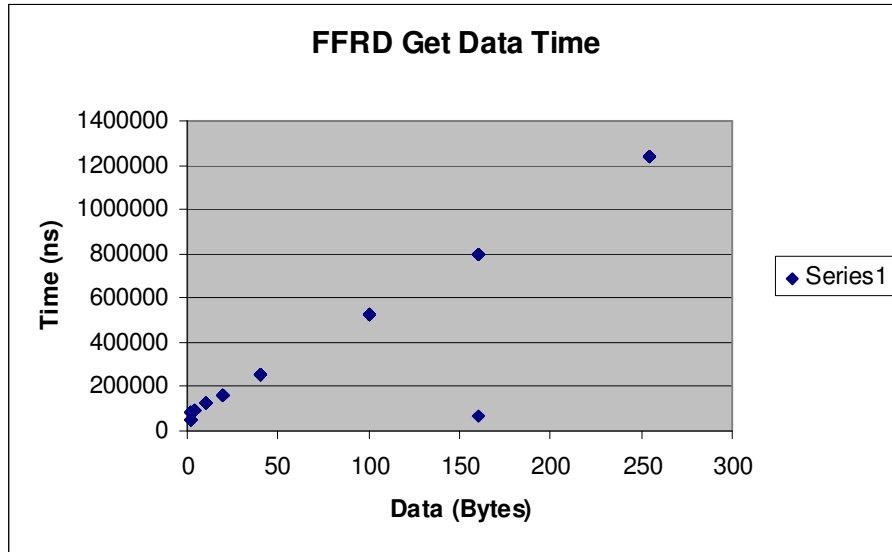


Figure 13.23: Fujitsu FlexRay Driver receive timings

There was an outlier contained within the data however. The data was therefore ‘cleaned’. To do this each set of timing figures were looked at. If a timing value was not consistent with the other values in its range then this value was removed. Interrupt latency could explain the outliers that were much larger than similar times. This is due to possible implementations of the E-Ray chip itself. The implementation may delay the signalling of an interrupt for an unknown time. The detection by the host of an interrupt can also not be accurately determined.

In terms of results that were obtained that were smaller than similar results, these times may be due to the communications controller losing synchronisation. For example when data is passed to the communications controller from the host, the data will not be passed if the controller is not synchronised to the bus. This will stop the driver function from completing a task and a small time for the execution will be recorded. It may also be due to the speed of the host detecting an interrupt event. This goes some way to explaining the outliers observed in the collected data.

Figure 13.24 shows a ‘cleaned’ data set where the outliers were taken out. This gives a better correlation between the payload size and the transfer time.

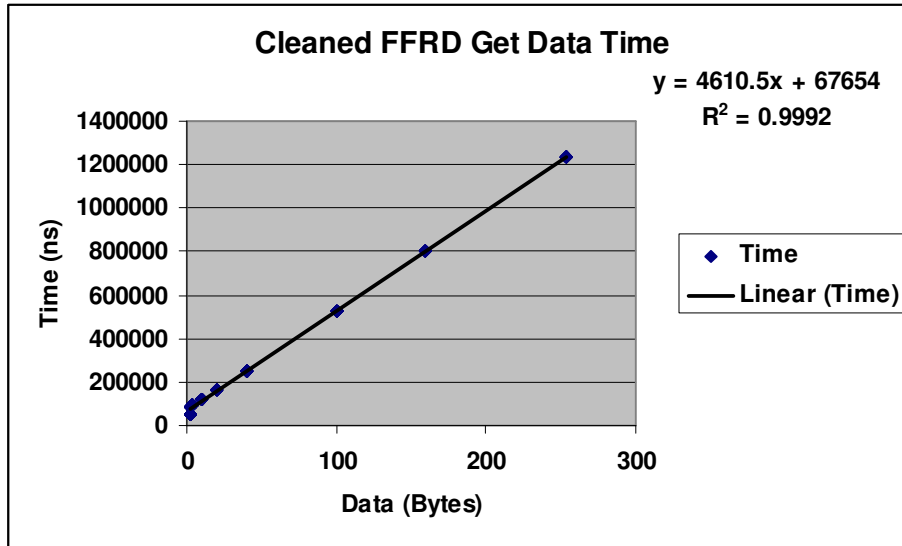


Figure 13.24: Fujitsu FlexRay Driver receive timings with linear trend line

As the linear trend line had the best correlation between the two sets of data, the trend line formula was taken as the relationship between the two sets of data.

13.5.1.4 COMMSTACK Transmit Times

The COMMSTACK software driver was tested next. It was again discovered that there is a strong relationship between the size of the data to be transferred and the time taken to transfer the data as can be seen in Figure 13.25 where one value varies with the other.

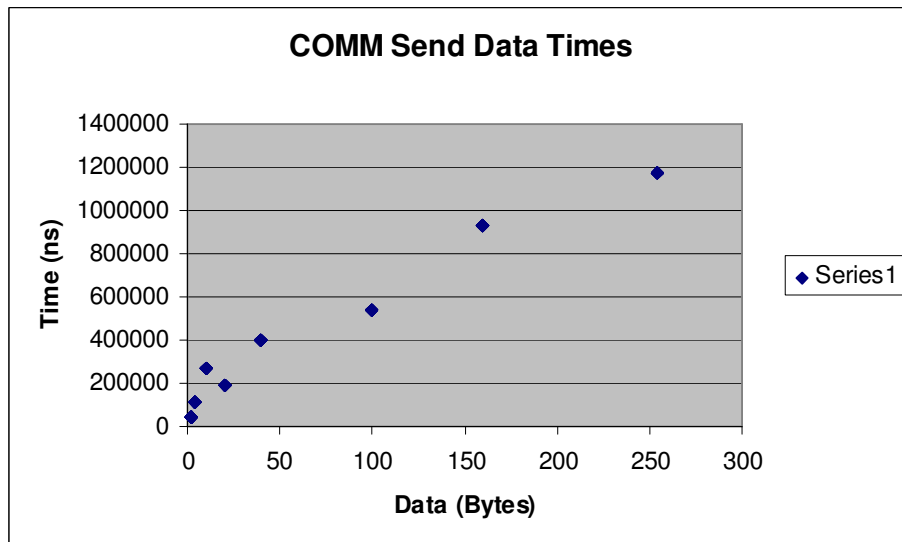


Figure 13.25: COMMSTACK transmit timings

MODEL CALIBRATION & VALIDATION

The correlation is similar to the Fujitsu FlexRay software driver but the polynomial trend line was a better fit. The two diagrams below, Figures 13.26 and 13.27, show the R^2 value for both a polynomial trend line and a linear trend line. This was done using a cleaned set of data.

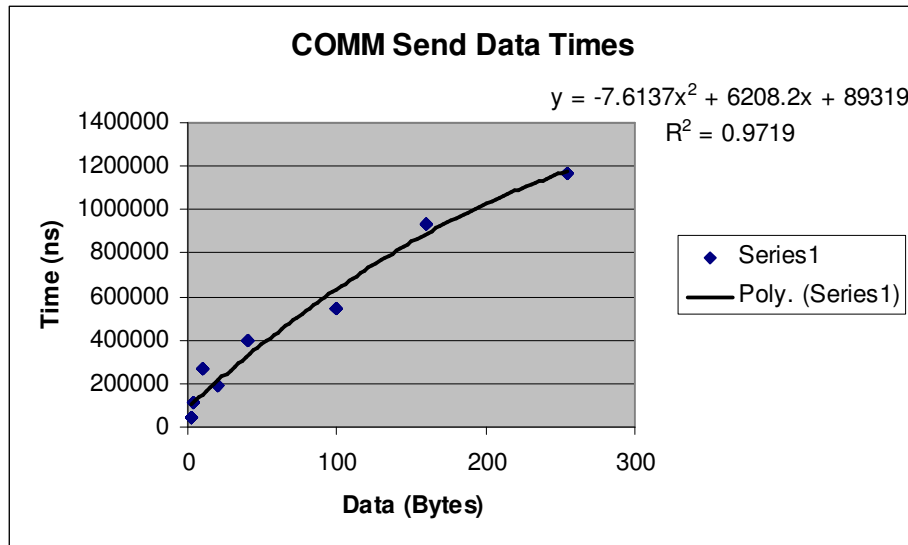


Figure 13.26: COMMSTACK transmit timings with polynomial trend line

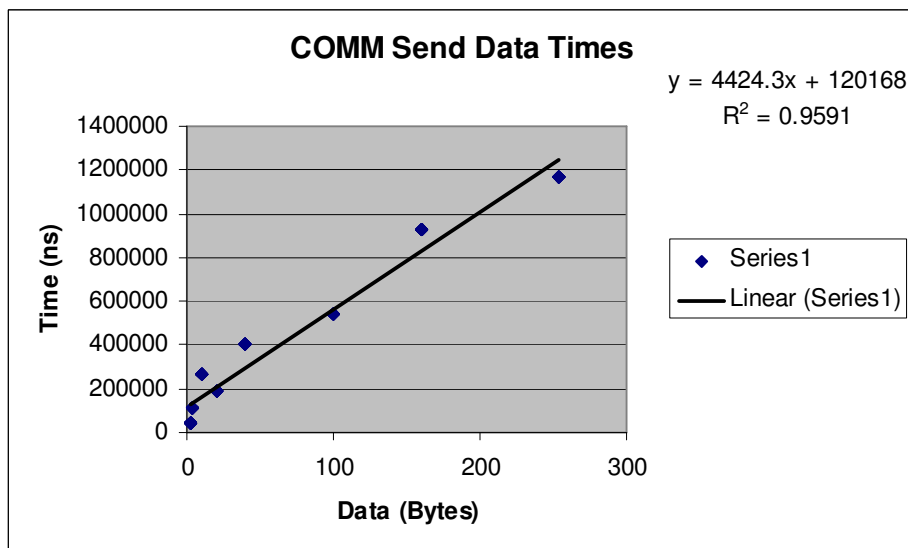


Figure 13.27: COMMSTACK transmit timings with linear trend line

Despite the Fujitsu FlexRay driver (FFRD) having a linear relationship it was decided to use the polynomial formula for the COMMSTACK data set. This is as the two software drivers may perform the sending of data to the communications controller

in different ways. As the polynomial trend line was a better fit in this case the formula was used to calculate the simulation models delays.

13.5.1.5 COMMSTACK Receive Times

In Figure 13.28 a strong relationship between the size of the data to be transferred and the time taken to transfer the data can be seen.

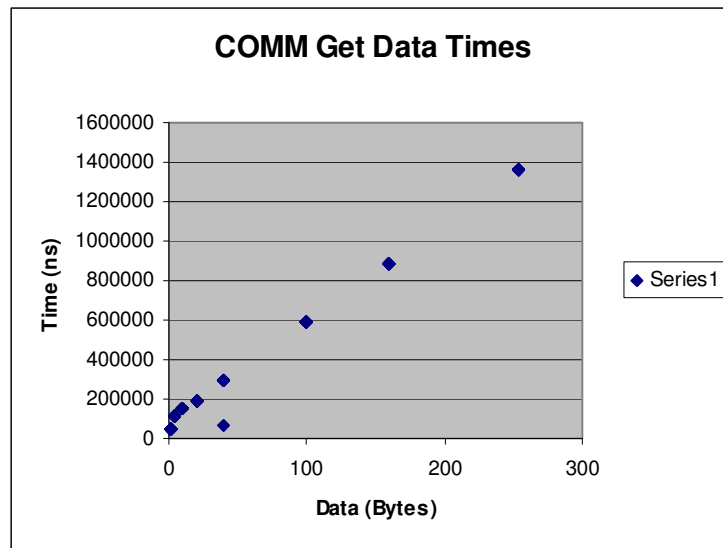


Figure 13.28: COMMSTACK receive timings

When the data was cleaned up the following trend line (as shown in the Figure 13.29) was discovered to have roughly the same fit as the polynomial trend line. The polynomial had an R^2 value of 0.9972 while the linear trend line formula had an R^2 value of 0.997. This was seen as a very minor difference. The COMMSTACK software driver will also to have be presented data from the communications controller in the same way the FFRD software driver is presented data. Therefore there would be minor differences in the handling of the data. It was therefore decided that the linear relationship would be used.

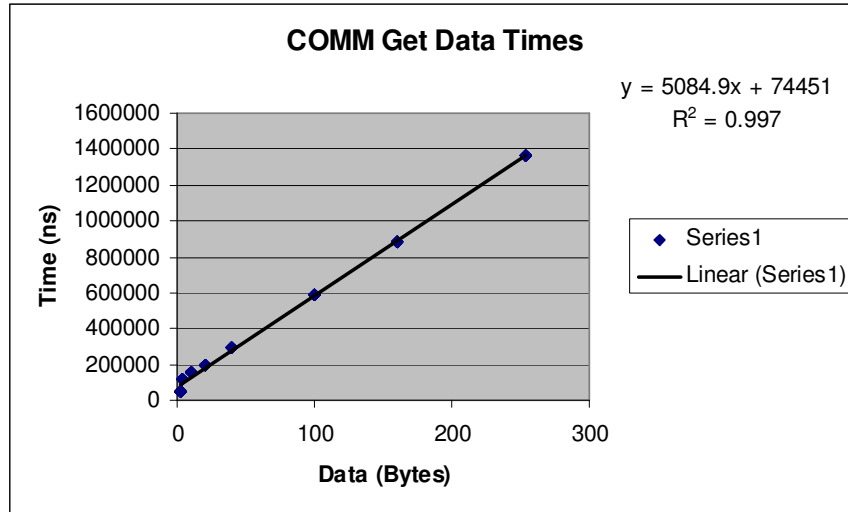


Figure 13.29: COMMSTACK receive timings with linear trend line

13.5.1.6 Transmit Timings

The transmit timings were obtained using a time stamp when the transmit frame interrupt was received by the host. This time stamp was compared to the expected slot start time and the expected transmission time length. The remaining time was then taken as the time it took to transfer the data between a transient buffer and the protocol controller block. Figure 13.30 shows the R^2 value and data points for the interrupt times collected.

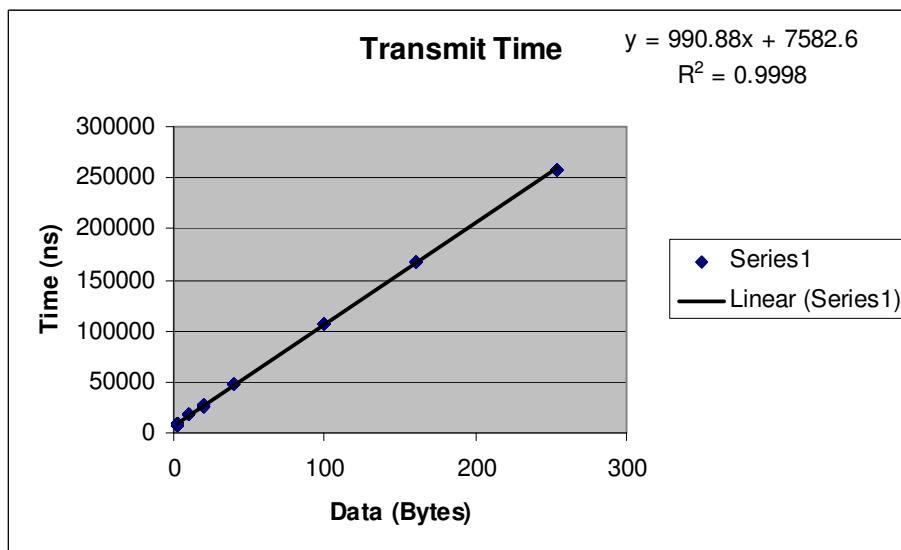


Figure 13.30: Transmit interrupt timing with linear trend line

MODEL CALIBRATION & VALIDATION

The data range was analysed by removing the calculated transmit times for the different payload lengths. This left a set of points as shown in Figure 13.31. The values of the transmit times are less than the frame transmission times. This indicates that the transmit interrupt occurs when a message has been successfully passed to the protocol transceiver from a transient buffer. This is justified as the interrupt is only generated when a frame is successfully transmitted. The interrupts can therefore not occur before this point. After a message is transferred to the transceiver the message can be transmitted. This must happen within a given time for message to be accepted as valid. The FlexRay protocol allows for this as well as allowing for time at the end of a slot. Almost all the times observed were all within this threshold. This is also the only time that can be deduced for this stage.

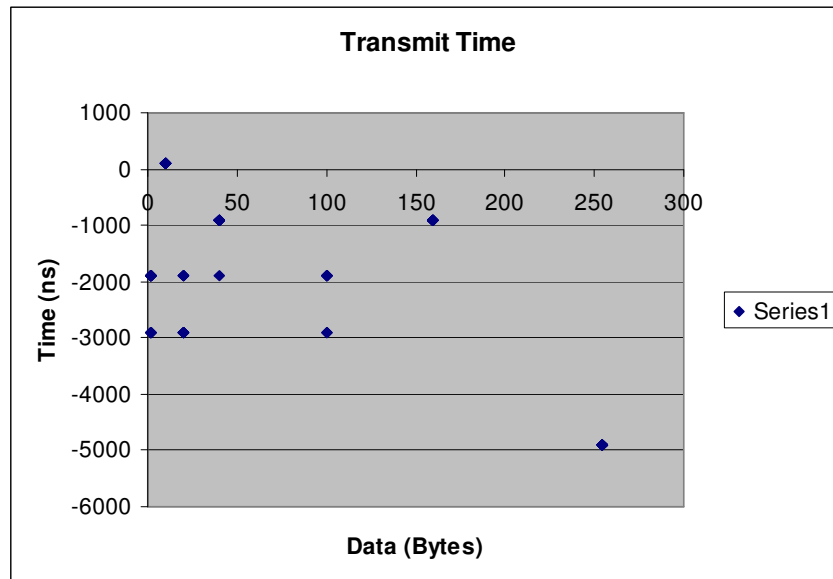


Figure 13.31: Transmit times

The transfer times obtained between the transient buffer and protocol transceiver are shown Figure 13.32 below. There are number of different points on the graph but there is no straight line on which all the points lie that can be determined from the information. Therefore the data was analysed for the averages of the information.

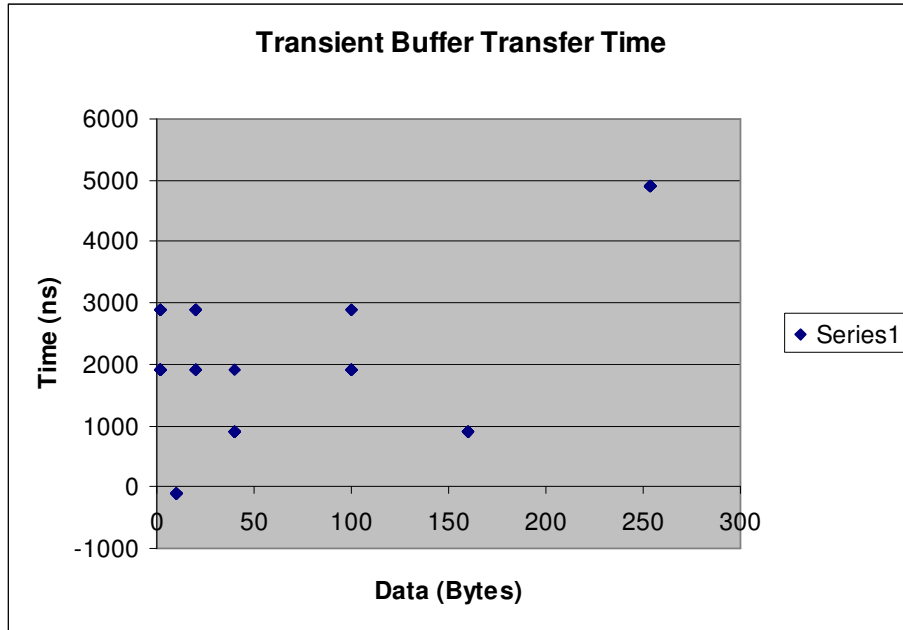


Figure 13.32: Transient buffer transfer times

Table 13.7 shows the mean, median and mode for the data set. All the values are a similar value (i.e. 1900 ns). The calculated mean is also greater than 1900 ns by less than 5%. For this reason the time for this stage is taken as 1900 ns. As there was no way to determine the time it takes to transfer a message from the protocol transceiver to the transient buffer, the value of 1900 ns was assumed for this time also.

	Time (ns)
mean	1987.5
median	1900
mode	1900

Table 13.7: Transmit averages

13.5.1.7 Receive Timings

The receive timings refer to the time taken for a frame assigned buffer to signal that a message is received. The time the buffer was updated was taken from an interrupt. This time was compared to the expected start time the relevant transmission slot and the time to transmit a frame. The time to transfer the message to the protocol transceiver and vice versa was also taken into account. This meant the time for the slot to be received and the buffer updated was then known. Figure 13.33 shows a graph of the receive interrupt times measured.

MODEL CALIBRATION & VALIDATION

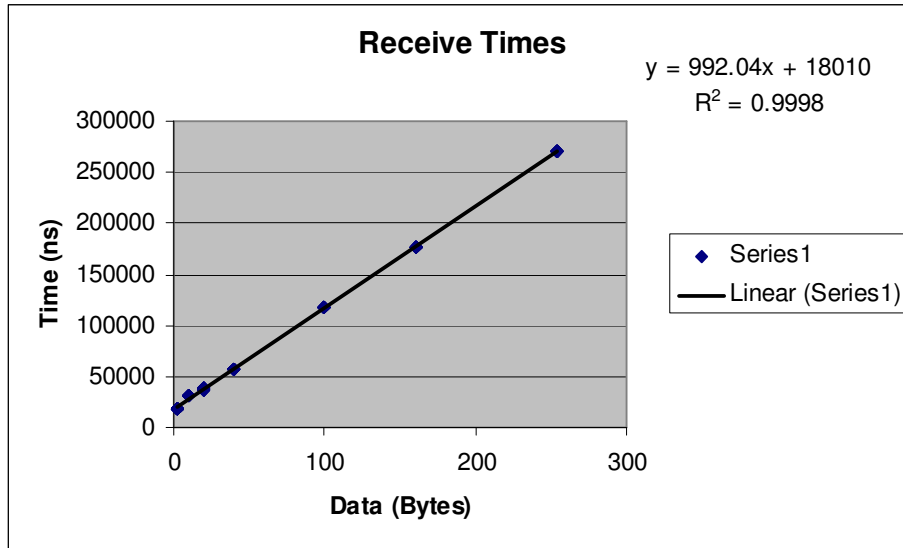


Figure 13.33: Receive interrupt timings

In Figure 13.34 the calculated buffer update times are shown. As with the transfer timings between a transient buffer and the protocol transceiver, the data is not very correlated. However as with the transient buffer and the protocol transceiver transfer times that was discussed in 13.5.1.6, the averages of this data set can be analysed.

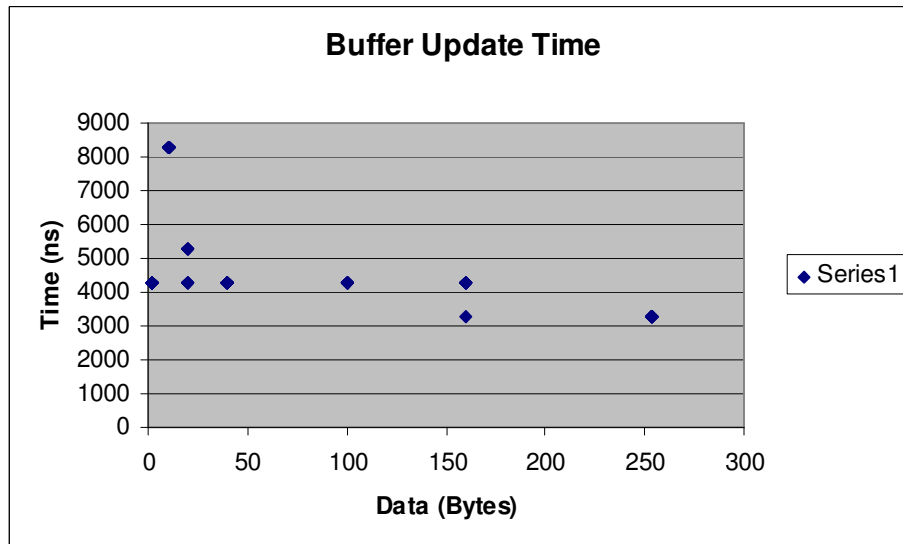


Figure 13.34: Buffer update timings

In Table 13.8 the calculated mean, median and mode for the data set can be seen. The median and mode were calculated as 4300 ns with the mean roughly 10% more at

MODEL CALIBRATION & VALIDATION

4725 ns. As the mean was only 10% greater than the median or mode the buffer update time was taken as 4300 ns. The time to read a Buffer was also taken as this time as there was no way to determine this time accurately.

	Full Data Range Time (ns)
mean	4725
median	4300
mode	4300

Table 13.8: Receive averages

The time of 4300 ns must also be taken as the buffer update time as the message handler simply selects which E-Ray block has access to the message RAM at any particular time. The time obtained is therefore a measure of the time to access a particular message buffer.

13.5.1.8 Input Buffer Transfer Timings

The interrupt time occurs at seemingly correlated times as the timer was started just before the FFRD driver sent the information to the communications controller. However the interrupt time does not represent the time it takes to transfer the data between the input buffer of the E-Ray chip and the message RAM. When the software drivers transfer time is taken into account the real input buffer transfer time can be calculated. It is also necessary to take into account the buffer access time calculated in section 13.5.1.7. Figure 13.35 shows the input buffer interrupt times minus the interrupt handler time and buffer update time.

MODEL CALIBRATION & VALIDATION

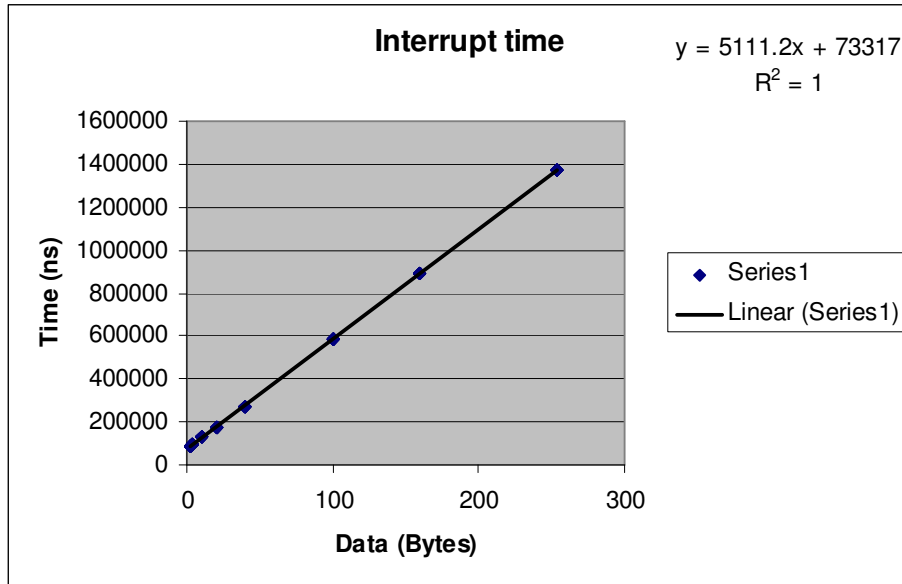


Figure 13.35: IBF interrupt timings with series trend line

When the software driver's times are accounted for, there is very low correlation between the transfer times. This is shown in Figure 13.36. The R^2 value obtained indicates there is a small correlation between the payload size and the transfer time.

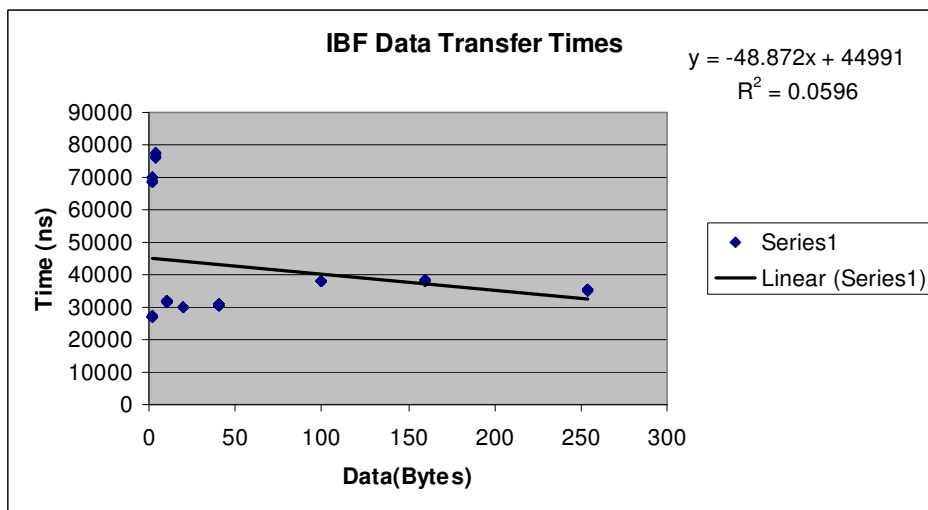


Figure 13.36: IBF timings with linear trend line

The mean, medium and mode for the data were therefore calculated for the data set. The calculated values are shown in Table 13.9. These averages are a measure of the time it takes a message to be passed from the input buffer and a message buffer in the message RAM. The median and mode values are both equal to 30250 ns. The mean of

MODEL CALIBRATION & VALIDATION

the collected data is larger at 41732.11 ns. However there are a number of outliers in the collected data. This could be due to the input buffer waiting to gain access to the message RAM. The time of 30250 ns was therefore chosen as the average input buffer time.

A buffer update time of 4300 ns was also already determined from the Receive timing data set. This would indicate that the average time spent in the input buffer is $30250 \text{ ns} - 4300 \text{ ns} = 25950 \text{ ns}$.

	Full Data Range Time (ns)
Mean	41735.119
Median	30250
Mode	30250

Table 13.9: IBF averages

13.5.1.9 Output Buffer transfer Timings

As with the IBF values the interrupt times are correlated until the FFRD time is taken into account. When the FFRD times are taken into account there is low correlation between the payload size and transfer time. Figure 13.37 shows the interrupt times with cleaned up data. The buffer read time that was calculated was also taken into account in the graph.

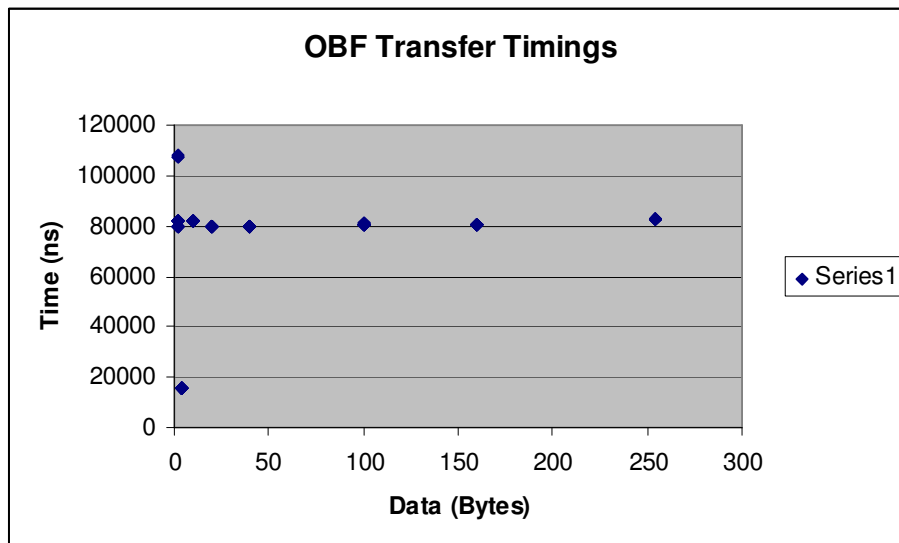


Figure 13.37: OBF interrupt timings

MODEL CALIBRATION & VALIDATION

The mean, median and mode were again calculated for this data set. Table 13.10 shows the averages calculated. The value of 79950 ns was taken as the average output buffer transfer time. Again the time to access a message buffer needs to be taken into account. This gives an overall output buffer transfer time average of 79950 ns - 4300 ns = 75650 ns

Full Data Range Time (ns)	
mean	74250.5952
median	79950
mode	79950

Table 13.10: OBF averages

When the analysis of this data was done it was realised that the software driver receive times calculated would be affected by these timings. The two software drivers receive times were then analysed to calculate correct and useable formule. Figures 13.38 and 13.39 show the amended software driver receive times.

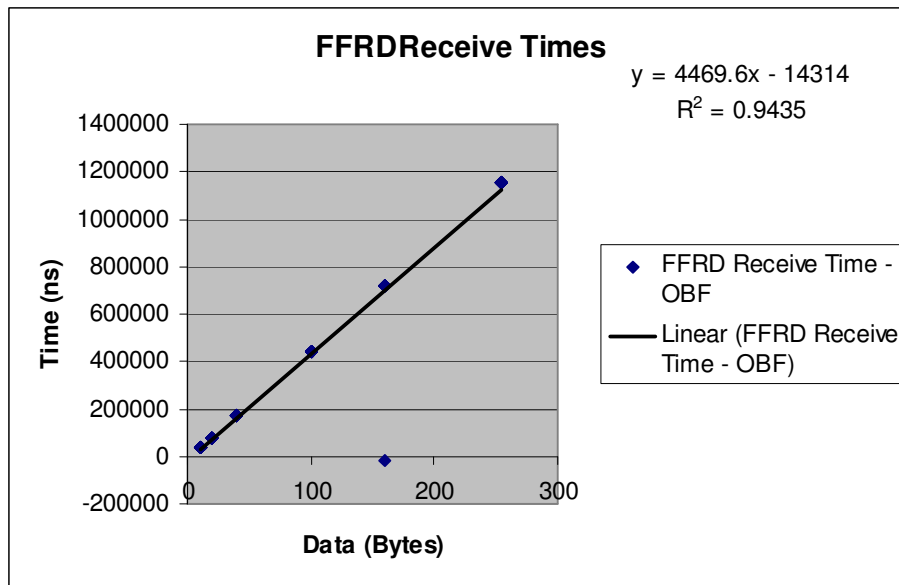


Figure 13.38: FFRD amended receive timings with linear trend line

MODEL CALIBRATION & VALIDATION

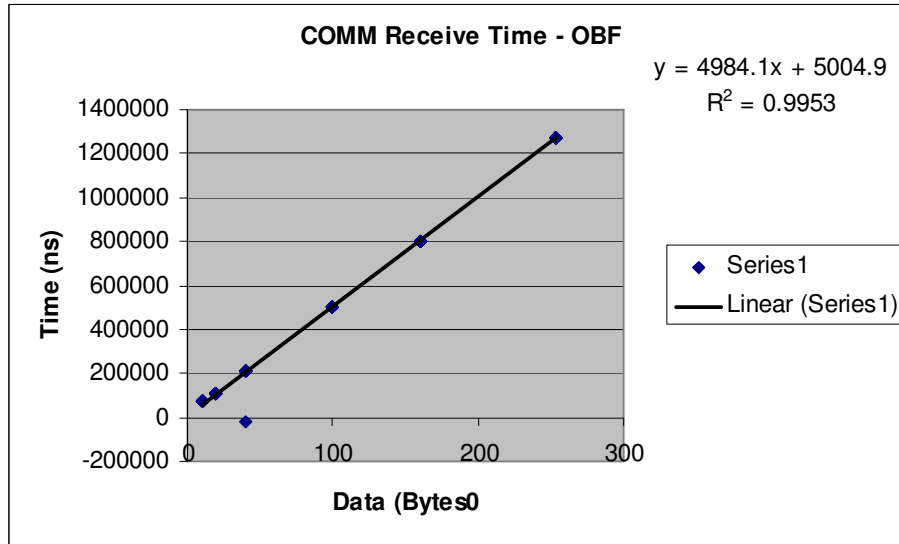


Figure 13.39: COMMSTACK amended timings with linear trend line

By using the formule obtained from this analysis, more confidence can be placed in the timing applied to the various points of the system between the message RAM and the application layer.

13.5.1.10 Discussion of the Data

The technique that was used involved using interrupts. This is not ideal as there is an overhead associated with an interrupt in terms of time. When the interrupt occurs it may not be serviced immediately if there is a higher or equal priority interrupt being serviced. All possible measurements were taken to reduce the effect of these times on the gathered data.

From the list of desired timing parameters given in section 13.5.1 it was known what timing information was received from the system. The time to transmit a frame of a given payload size was calculated based on the FlexRay specifications. It was necessary to monitor the rest of the values. The data collected was then carefully analysed for correct timing relationships. The timing data for the Fujitsu FlexRay software driver and the COMMSTACK software driver were both obtained. For the purpose of the calibration procedure however the model will only be calibrated to the COMMSTACK software driver.

All the timing data collected will not be a completely accurate representation of the timings of a real world system. With the time and equipment available however this is the best data that can be obtained. The calibration of the model will therefore be

MODEL CALIBRATION & VALIDATION

conducted against the best possible data set available. The timing value and formula for each parameter are shown in Table 13.11. In Table 13.11 'x' represents the data size of any entity passing through that layer. These timing constraints can then be applied to the simulation model. The simulation model can then be monitored and the timing data obtained can be compared to the actual real world system.

Constraint	Time (ns)
COMMSTACK Transmit Time	$(-7.6137 * x^2) + (6208.2 * x) + 89319$
COMMSTACK Receive Time	$(4984.1 * x) + 5004.9$
Input Buffer Handling Time	25950
Output Buffer Handling Time	75650
Message RAM Access Time	4300
Transient Buffer Handling Time	1900 ns
Physical Bus Transfer Time	$((6 + 1 + 2 + (8 * (8 + x)) + (2 * (x + 8))) * 100)$

Table 13.11: System timing constraints

The timing data for the FIFO was not obtainable due to time constraints and errors when collecting the data. Unfortunately this meant that Test Case 6 could not be successfully completed. The calibration tests could only be carried out over the first five test cases as there was no FIFO timing to compare the data from the model to. This is due to the fact the Designer configuration software does not at the present time support FIFO applications.

13.5.2 Calibration Verification Results

The model for each test case was configured with the timing constraints and the tests were run. The model needed to be configured to obtain all appropriate data however. Each of the tests was originally run over a single communication cycle. However this did not always produce the desired data. This is because the random element of deciding the generation of frames on the physical bus will not always produce the frames for the simulation node to store. Also the updating of a transmit message buffer may not happen before the buffer is checked for any transmit data. In order to check the model acted in the desired manner the model tests needed to be changed.

MODEL CALIBRATION & VALIDATION

The possibility to run the test over two communication cycles was explored. This would guarantee that all frames would be transmitted from the simulated node at least once. However there was still no guarantee that the desired frames would be generated by the physical bus subsection. The model was therefore changed to generate only those frames that were desired for the tests. The random element of the physical bus was also removed for these tests to ensure that a frame entity would be generated for desired slots. To ensure the node model transmitted the desired frames during a single communication cycle, the message RAM buffer entities were initialised to indicate all relevant messages buffers were ready for transmission. This can be seen as the buffer having been updated at a time before the simulation start time. This will ensure that any buffer that is checked before it is updated will still transmit the associated frame.

Other changes made to the model were that only a single request was generated by the application layer model subsystem at the start of a communication cycle. Other requests are then generated when a request has signalled completion. This will allow the application to act in a more realistic manner when compared to the real world system. The final change involved the timing constraints and the implementation into the model. The majority of the timing constraints were fixed lengths and these were simple set as the time of a single server block. To implement the software driver and physical bus timings lookup tables were used. Lookup tables had been used in other sections of the model and were verified as being suitable to produce the correct output for a desired input. Tables 13.12 – 13.16 compare the timing obtained from the simulation model and the timing constraints imposed on them. However the tests outlined in this section, section 13.5.2, were run to verify that the timing obtained from the model matched the constraints as calculated in sections 13.5.1.

MODEL CALIBRATION & VALIDATION

Test Case: 1	Calculated Constraint (ns)		Simulation Model Time (ns)	
Software Driver Transmit Time	Frame3	101704.9	Frame3	101704.9
	Frame44	101704.9	Frame44	101704.9
Software Driver Receive Time	Frame6	141973.1	Frame6	14973.1
	Frame18	141973.1	Frame18	14973.1
Input Buffer Time	Frame3	25950	Frame3	25950
	Frame44	25950	Frame44	25950
Output Buffer Time	Frame6	75650	Frame6	75650
	Frame18	75650	Frame18	75650
Media Access Buffer Time	Frame3	1900	Frame3	1900
	Frame44	1900	Frame44	1900
Frame and Symbol Processing Time	Frame6	1900	Frame6	1900
	Frame18	1900	Frame18	1900
Physical Bus Timing	Frame3	10900	Frame3	10900
	Frame44	10900	Frame44	10900
	Frame6	10900	Frame6	10900
	Frame18	10900	Frame18	10900
Note: All model subsystems timing data was calibrated to the desired times.				

Table 13.12: Calibration test case 1 data

Test Case: 2	Calculated Constraint (ns)		Simulation Model Time (ns)	
Software Driver Transmit Time	Frame2	210437.5	Frame2	210437.52
	Frame444	325465.05	Frame444	325465.08
Software Driver Receive Time	Frame1	104686.9	Frame1	104686.9
	Frame181	204368.9	Frame181	204368.9
Input Buffer Time	Frame2	25950	Frame2	25950
	Frame444	25950	Frame444	25950
Output Buffer Time	Frame1	75650	Frame1	75650
	Frame181	75650	Frame181	75650
Frame Transmit Time	Frame2	1900	Frame2	1900
	Frame444	1900	Frame444	1900
Frame Receive Time	Frame1	1900	Frame1	1900
	Frame181	1900	Frame181	1900
Physical Bus Timing	Frame2	28900	Frame2	28900
	Frame444	48900	Frame444	48900
	Frame1	28900	Frame1	28900
	Frame181	48900	Frame181	48900
Note: All model subsystems timing data was calibrated to the desired times.				

Table 13.13: Calibration test case 2 data

MODEL CALIBRATION & VALIDATION

Test Case: 3	Calculated Constraint (ns)		Simulation Model Time (ns)	
Software Driver Transmit Time	Frame3	634002	Frame3	634002
	Frame65	887720.28	Frame65	887720.28
Software Driver Receive Time	Frame6	503414.9	Frame6	503414.9
	Frame66	802460.9	Frame66	802460.9
Input Buffer Time	Frame3	25950	Frame3	25950
	Frame65	25950	Frame65	25950
Output Buffer Time	Frame6	75650	Frame6	75650
	Frame66	75650	Frame66	75650
Media Access Buffer Time	Frame3	1900	Frame3	1900
	Frame65	1900	Frame65	1900
Frame and Symbol Processing Time	Frame6	1900	Frame6	1900
	Frame66	1900	Frame66	1900
Physical Bus Timing	Frame3	108900	Frame3	108900
	Frame65	168900	Frame65	168900
	Frame6	108900	Frame6	108900
	Frame66	168900	Frame66	168900
Note: All model subsystems timing data was calibrated to the desired times.				

Table 13.14: Calibration test case 3 data

Test Case: 4	Calculated Constraint (ns)		Simulation Model Time (ns)	
Software Driver Transmit Time	Frame3	1174996.331	Frame3	1174996.331
	Frame28	150639.63	Frame28	150639.63
Software Driver Receive Time	Frame6	1270966.3	Frame6	1270966.3
	Frame29	54845.9	Frame29	54845.9
Input Buffer Time	Frame3	25950	Frame3	25950
	Frame28	25950	Frame28	25950
Output Buffer Time	Frame6	75650	Frame6	75650
	Frame29	75650	Frame29	75650
Media Access Buffer Time	Frame3	1900	Frame3	1900
	Frame28	1900	Frame28	1900
Frame and Symbol Processing Time	Frame6	1900	Frame6	1900
	Frame29	1900	Frame29	1900
Physical Bus Timing	Frame3	262900	Frame3	262900
	Frame28	18900	Frame28	18900
	Frame6	262900	Frame6	262900
	Frame29	18900	Frame29	18900
Note: All model subsystems timing data was calibrated to the desired times.				

Table 13.15: Calibration test case 4 data

MODEL CALIBRATION & VALIDATION

Test Case: 5	Calculated Constraint (ns)		Simulation Model Time (ns)	
Software Driver Transmit Time	Frame2	114029.9808	Frame2	114029.9808
Software Driver Receive Time	Frame1	24941.3	Frame1	24941.3
Input Buffer Time	Frame2	25950	Frame2	25950
Output Buffer Time	Frame1	75650	Frame1	75650
Media Access Buffer Time	Frame2	1900	Frame2	1900
Frame and Symbol Processing Time	Frame1	1900	Frame1	1900
Physical Bus Timing	Frame2	12900	Frame2	12900
	Frame1	12900	Frame1	12900
Note: All model subsystems timing data was calibrated to the desired times. To obtain all results, the model was run for twice. The first test requested to update the buffer for frame 2. The second test requested the data stored in the buffer for frame 1 only.				

Table 13.16: Calibration test case 5 data

For each of the test cases the model followed the desired pattern of timing constraints. The use of look up tables ensures that the correct timing delay is chosen where a variable delay is associated. They also reduces the execution time of the model by reducing the number of calculations at run time. The models ability to calculate a desired subsystem delay is therefore verified as working as desired.

13.5.3 Calibration Results vs. Real World Results

As the timing of the model was verified as working, the models timing output must be compared to real world system timing. This will give a measure of how well the model was calibrated. To do this the data from the model was again analysed to produce

MODEL CALIBRATION & VALIDATION

a set of timing data that is a match to the real world system timings. Tables 13.17-13.21 show the calibration data.

All real world values are averages of the collected data set for each message identifier. All the real world values are the time related to analysed subsystem data. For instance the frame receive and transmit times are based on the time of the interrupt compared to the start of the associated slot start time. The input buffer and output buffer times relate to the time for information to pass through these stages only. All these times also take into account the interrupt latency time. The correlation column on the right of Tables 13.17-13.21 relates how well the model data set corresponds to the real world data set.

A 10% divergence between the real world results and the simulation results was deemed acceptable after an investigation into the timing constraints of FlexRay was undertaken (see Appendix B.2.1 of the FlexRay specifications). It was found that the greatest static slot action point offset was 63 macroticks and the longest duration of a static slot is 661 macroticks. This means that for valid communication to be achieved with the largest static slot that could be defined, a maximum of approximately 10% of the slot could be taken as an empty space, (i.e where no communication appears on the physical bus), before the frame transmission begins. A 10% difference was then used as a measure for the acceptable divergence value for each of the calibration and validation tests as outlined in this thesis.

Test Case 1	Real World Time (ns)		Simulation Model Time (ns)		Divergence (%)
	Frame	Time (ns)	Frame	Time (ns)	
Software Driver Transmit Time	Frame3	111750	Frame3	101704.9452	-8.99
	Frame44	111475	Frame44	101704.9452	-8.76
Software Driver Receive Time	Frame6	32585.7	Frame6	14973.1	-54.05
	Frame18	21571.4	Frame18	14973.1	-30.59
Input Buffer Time	Frame3	21527.8	Frame3	30250	+40.52
	Frame44	21277.8	Frame44	30250	+42.17
Output Buffer Time	Frame6	93416.6667	Frame6	79950	-14.42
	Frame18	86055.5556	Frame18	79950	-7.10
Frame Transmit Time	Frame3	19000	Frame3	12800	-32.64
	Frame44	19000	Frame44	12800	-32.64
Frame Receive Time	Frame6	19000	Frame6	19000	0.00
	Frame18	19000	Frame18	19000	0.00

Table 13.17: Calibration test case 1 analysis

MODEL CALIBRATION & VALIDATION

Test Case 1 Results:

From the correlation column there are a number of values close to $\pm 10\%$. It can be seen that there are a number of values that range from approximately -54% up to $+42\%$. These values will not produce an accurate representation of the data flow around the FlexRay node.

Test Case 2	Real World Time (ns)		Simulation Model Time (ns)		Divergence (%)
Software Driver Transmit Time	Frame2	189650	Frame2	210437.5	+10.96
	Frame444	399700	Frame444	325465.1	-18.58
Software Driver Receive Time	Frame1	120933.3	Frame1	104686.9	-13.43
	Frame181	204968.8	Frame181	204368.9	-0.29
Input Buffer Time	Frame2	31750	Frame2	30250	-4.72
	Frame444	32277.8	Frame444	30250	-6.28
Output Buffer Time	Frame1	84222.2222	Frame1	79950	-5.08
	Frame181	84250	Frame181	79950	-5.11
Frame Transmit Time	Frame2	26400	Frame2	28900	+9.46
	Frame444	47900	Frame444	50800	+6.05
Frame Receive Time	Frame1	37500	Frame1	35101	-6.40
	Frame181	57000	Frame181	55101	-3.34

Table 13.18: Calibration test case 2 analysis

Test Case 2 Results:

The data for this calibration test case is a lot closer than that of test case 1. The majority of values lie within $\pm 10\%$ of the real world values. Again this is deemed acceptable for the initial calibration run. However there are two values that are almost -20% off the real world value. These times may again produce inaccurate results.

MODEL CALIBRATION & VALIDATION

Test Case 3	Real World Time (ns)		Simulation Model Time (ns)		Divergence (%)
	Frame	Time (ns)	Frame	Time (ns)	
Software Driver Transmit Time	Frame3	544525	Frame3	634002	+16.43
	Frame65	931650	Frame65	887720.3	-4.72
Software Driver Receive Time	Frame6	503666.667	Frame6	503414.9	-0.05
	Frame66	800666.667	Frame66	802460.9	+0.22
Input Buffer Time	Frame3	38055.56	Frame3	30250	-20.52
	Frame65	38277.78	Frame65	30250	-20.97
Output Buffer Time	Frame6	85027.7778	Frame6	79950	-5.97
	Frame66	85000	Frame66	79950	-5.94
Frame Transmit Time	Frame3	106700	Frame3	110800	+3.84
	Frame65	168000	Frame65	170800	+1.67
Frame Receive Time	Frame6	117000	Frame6	115101	-1.62
	Frame66	176900	Frame66	175101	-1.02

Table 13.19: Calibration test case 3 analysis

Test Case 3 Results:

Like the data of test case 2, the majority of values lie within $\pm 10\%$ of the real world values. Again this is deemed acceptable for the initial calibration run. However there are two values that are almost -20% off the real world value. These times may again produce inaccurate results.

Test Case 4	Real World Time (ns)		Simulation Model Time (ns)		Divergence (%)
	Frame	Time (ns)	Frame	Time (ns)	
Software Driver Transmit Time	Frame3	1171875	Frame3	1174996	+0.27
	Frame28	267950	Frame28	150639.6	-43.79
Software Driver Receive Time	Frame6	1274175	Frame6	1270966.3	-0.25
	Frame29	70125	Frame29	54845.9	-21.79
Input Buffer Time	Frame3	35166.7	Frame3	30250	-13.98
	Frame28	31944.4	Frame28	30250	-5.30
Output Buffer Time	Frame6	87000	Frame6	79950	-8.10
	Frame29	86500	Frame29	79950	-7.57
Frame Transmit Time	Frame3	258000	Frame3	264800	+2.64
	Frame28	19000	Frame28	20800	+9.47
Frame Receive Time	Frame6	270000	Frame6	271000	+0.37
	Frame29	31000	Frame29	27000	-12.90

Table 13.20: Calibration test case 4 analysis

MODEL CALIBRATION & VALIDATION

Test Case 4 Results:

The data for test case 4 follows the pattern of the previous 3 test case data sets. The majority of the results lie within $\pm 10\%$ of the real world values but with four values outside this range.

Test Case 5	Real World Time (ns)		Simulation Model Time (ns)		Divergence (%)
Software Driver Transmit Time	Frame2	116850	Frame2	114030	-2.41
Software Driver Receive Time	Frame1	30850	Frame1	24941.3	-19.16
Input Buffer Time	Frame2	34600	Frame2	30250	-12.57
Output Buffer Time	Frame1	84150	Frame1	75650	-10.10
Frame Transmit Time	Frame1	9000	Frame1	12901	+43.34
Frame Receive Time	Frame2	20000	Frame2	19101	-4.49

Table 13.21: Calibration test case 5 analysis

Test Case 5 Results:

There is one value that is approximately 43% greater than the desired value. There are then three values that are in an around 10% different while two more values lie within 20% of their desired values.

13.5.3.1 Calibration Results vs. Real World Conclusion

The first run of calibration tests produced a set of data that was then compared to the real world values. The number of values that were outside a difference of $\pm 10\%$ compared to the real world system was recorded. It was decided that a difference of $\pm 10\%$ was a reasonable level of accuracy for an calibration run. When the model is within $\pm 5\%$ of the real model reasonable assumptions of the system based on the model output can be made. To increase the accuracy of the model will require the study of the areas where the model underperforms and it is therefore unlikely that a single run of calibration tests will produce a highly accurate model. If the accuracy of a model is below $\pm 10\%$ however there is likely to be some error in the implementation of the model. This could be in the implementation of the model subsystems or in the collected

MODEL CALIBRATION & VALIDATION

timing data from the real world system. Table 13.22 shows the number of these ‘errors’ for each timing parameter.

Parameters	Unacceptable Differences
Software Driver Transmit Time	3
Software Driver Receive Time	6
Input Buffer Time	5
Output Buffer Time	1
Frame Transmit Time	3
Frame Receive Time	1

Table 13.22: Calibration test results summary

From Table 13.22 it is clear that the greatest errors were observed in the ‘Software Driver Receive’ and ‘Input Buffer’ Times. It is possible that by concentrating on improving these values the model can achieve an acceptable level of accuracy.

The timing of the model follows the applied constraints perfectly as can be seen in section 13.5.2. One way to calibrate the model more accurately would be to collect more real world data. This could therefore mean that there was insufficient calibration test cases performed to accurately calibrate the model. It may also be possible that with a limited number of usable interrupts, on the E-Ray communications controller, that insufficient timing data can be obtained in this way. More data, from more calibration test runs, will allow an increase in the understanding of the timing constraints and may be the most practical solution to increase the model accuracy. This new data can then again be applied to the model and the output checked. Another possibility would be to look at how the model was constructed and behaves. It could be discovered from analysis of the model that a subsystem of the simulation does not accurately reflect the real-world implementation. The subsystem could then be re-modelled by using different modelling blocks to achieve the desired output from the system. This could then be adjusted if needed to develop a more accurate simulation model. It is suspected that the message handler subsystem of the simulation model may not handle the various entities passed to it in most suitable manner. Further calibration runs should therefore concentrate on this area of the simulation model.

This initial calibration run has identified where the simulation model differs from the real world system. More calibration runs can help improve the accuracy of the

system to a 95 or 99% accuracy. The accuracy desired will determine the number of runs ultimately needed.

13.5.4 Data Pipeline Analysis

The model was designed to analyse the flow of data through the system. The model was therefore constructed as two pipelines. The data either originates from the application layer, is sent through the pipeline to the physical bus, or the data originates from the physical bus up through the receive pipeline to the application layer. This means that the overall timing of the pipelines will give a measure of the accuracy of the simulation model as a whole and not just each subsystem.

To ensure that the pipeline timing is accurate the overall flow of data was analysed. The transmitted data originates from the application layer, passes through the software driver, input buffer and communications controller. The overall path time for the real world system and the simulation model were compare as shown in Table 13.23. Again the correlation column shows how well the two sets of data relate to each other. The difference between the two sets of data is also compared to the static slot time length as well as the communication cycle. This produces a good accuracy metric.

MODEL CALIBRATION & VALIDATION

Transmit Pipeline	Real World Time (ns)	Simulation Model Time (ns)	Divergence (%)	Slot Time Correlation (%)	Cycle Time Correlation (%)		
Test Case 1	Frame 3	152277.8	Frame 3	144754.9452	-4.94	30.09	0.05
	Frame 44	151752.8	Frame 44	144754.9452	-4.61	27.99	0.04
Test Case 2	Frame 2	247800	Frame 2	269587	+8.79	50.67	0.14
	Frame 444	479877.8	Frame 444	406515.1	-15.29	170.61	0.46
Test Case 3	Frame 3	689280.56	Frame 3	775052	+12.44	69.73	1.72
	Frame 65	1137927.78	Frame 65	1088770.3	-4.32	39.97	0.98
Test Case 4	Frame 3	1465041.7	Frame 3	1470046	+0.34	1.80	0.10
	Frame 28	318894.4	Frame 28	201689.6	-36.75	42.16	2.34
Test Case 5	Frame 2	160450	Frame 2	157181	-2.04	12.11	2.87
Average Correlation					-5.15	49.46	0.97

Table 13.23: Transmit pipeline timing

The correlation between the transmit pipeline data for the real world and simulation model have a number of test cases that have a $\pm 10\%$ difference. Test case 2 and test case 4 both fall outside this. This means that for a majority of the test cases the model is within an acceptable range. When the differences are compared to the communication cycle length, the model has an accuracy value well within the various values (an accuracy of less than 100% percent indicates the model is accurate to within that parameter). The model can therefore be said to be accurate to within a cycle length. The differences were also checked against a static slot length. The accuracy for all but one test case fell within one static slot length (170.61 % accurate). Frame 444 of test case 2 had an accuracy of greater than one slot length, but less than two slot lengths. The accuracy of the transmit pipeline of the model can therefore be said to be accurate to within two static slot lengths.

MODEL CALIBRATION & VALIDATION

The received data originates from the physical bus, passes through the communications controller, input buffer and software driver. The overall path time for the real world system and the simulation model were compare as shown in Table 13.24.

Receive Pipeline	Real World Time		Simulation Model		Divergence (%)	Slot Time Correlation (%)	Cycle Time Correlation (%)
		(ns)	Time (ns)				
Test Case 1	Frame 6	127902.3667	Frame 6	113923.1	-10.93	55.91707	0.08737
	Frame 18	126626.9556	Frame 18	113923.1	-10.03	50.81542	0.079399
Test Case 2	Frame 1	242655.5222	Frame 1	219737.9	-9.44	53.2968	0.143235
	Frame 181	346218.8	Frame 181	339419.9	-1.96	15.8114	0.042493
Test Case 3	Frame 6	705694.4448	Frame 6	626465.9	-11.23	64.41345	1.584571
	Frame 66	1062566.667	Frame 66	1057511.9	-0.48	4.109567	0.101095
Test Case 4	Frame 6	1631175	Frame 6	1627916.3	-0.57	1.172194	0.065174
	Frame 29	187625	Frame 29	161795.9	-13.77	9.291043	0.516582
Test Case 5	Frame 1	135000	Frame 1	119692.3	-11.34	56.69519	13.42781
Average Correlation					-7.75	1.783081	34.61357

Table 13.24: Receive pipeline timing

The correlation between the receive pipeline data for the real world and simulation model have a number of test cases that have a $\pm 10\%$ difference. Test Case 2 and test case 4 both fall outside this. This means that for a majority of the test cases the model is within an acceptable range.

When the differences are compared to the communication cycle length, the model has an accuracy value well within the cycle length values. The model can therefore said to be accurate to within a cycle length. The differences were also checked against a static slot length. The accuracy for all of the test cases fell within one static slot length. The accuracy receive pipeline can therefore said to be accurate to within one static slot lengths.

13.6 Conclusion

The hardware and software tools described in this chapter have been chosen for various reasons. Some of the equipment was available to the Automotive Control Group at Waterford Institute of Technology (W.I.T.) before this project was started. Other equipment had to be bought from a set budget. As some of the equipment was already available to the research group it makes sense to use it. This means the budget can be used more efficiently.

The equipment that has been outlined is sufficient to implement the simulation of a FlexRay network as well as calibrate and validate the simulation model. The software that has been acquired has been done so with ease of use in mind amongst other considerations. This will mean for example less time will be spent getting familiar with the tools and so more time can be dedicated to the implementation of the project. The software was also chosen as it is used in industry and so is seen as reliable for the functions intended. The hardware that was chosen was done so as it too is easy to get familiar with. It was also chosen because it was industry standard and had a high level of reliability and performance.

The tests that were run were designed to obtain the real world timing data as accurately as possible. However there will be slight inaccuracies in the values obtained. This is down to a number of factors such as the limitation of the knowledge of actual implementations of a system. The model was calibrated to this data set as this was the best that could be obtained.

The calibration of the model outlined above obtained two sets of data. The simulation was then run and timing information obtained from the model. This was checked against the obtained real world timing constraints. The pipeline timing aspect of the model was also analysed and metrics of performance developed. Some of the values were inconsistent with the overall timing of the flow of data but may have been accurate enough to conduct initial validation testing. As the pipeline times were within acceptable times the calibration was deemed sufficient to perform the validation tests. In this way the initial calibration testing of the model was deemed a success. The simulation model is not calibrated to a high level of accuracy and further calibration test will be needed until an acceptable level of accuracy is achieved.

13.7 References

Banks, J., Carson, J. S., Nelson, B. L. and Nicol, D. M. (2001) Discrete-Event System Simulation, New Jersey: Prentice Hall.

Robert Bosch GmbH (2006) E-Ray FlexRay IP-Module User's Manual, Revision 1.2.3, Reutlingen: Robert Bosch GmbH.

CMP Media LLC (2007) Active Hub [online], available at:
<http://www.techweb.com/encyclopedia/defineterm.jhtml?term=activehub> [accessed 14 January 2008].

Dependable Computer Systems (2006) COMMSTACK <FlexRay> 1.8 User's Manual, Vienna, Austria.

Dependable Computer Systems (2007) DESIGNER PRO 4.3.0 - DESIGNER PRO, DESIGNER PRO <LIGHT> and DESIGNER PRO <SYSTEM> Document Version 2.2, Vienna, Austria.

Freund, J. (1979) Modern Elementary Statistics, 5th edition, New Jersey: Prentice Hall.

Fujitsu Limited (2004a) FR Family SOFTUNETM Workbench User's Manual for V6, Japan.

Fujitsu Limited (2004b) FR60 32-Bit microcontroller MB91460 Series User's Manual, Version 1.21, Japan.

Fujitsu Microcontroller Info Team (2008) R Re: [FME#2008081513000241] FlexRay examples, email to Robert Shaw (rshaw@wit.ie), 8 October [accessed 3 February 2009].

Fujitsu Microelectronics Europe (2007a) MB88121 Series, MB91460 Series Evaluation Board SK-91F467-Flexray Software Guide v1.5, Langen, Germany.

Fujitsu Microelectronics Europe (2007b) MB88121 Series, MB91460 Series Evaluation Board SK-91F467-Flexray User Guide v1.6, Langen, Germany.

MODEL CALIBRATION & VALIDATION

Fujitsu Microelectronics Europe (2007c) SK-91F467-FLEXRAY [online], available at: http://mcu.emea.fujitsu.com/mcu_tool/detail/SK-91F467-FLEXRAY.htm [accessed 24 January 2008].

Fujitsu Microelectronics Europe (2007d) Fujitsu FlexRay Driver Manual v1.3, Langen, Germany.

Fujitsu Microelectronics Europe GmbH (2007e) 91460_dynamic1_91467d-v16 [online], available at: http://mcu.emea.fujitsu.com/mcu_tool/detail/SK-91F467-FLEXRAY.htm [accessed 3 February 2008].

Fujitsu Microelectronics Europe GmbH (2007f) 91460_dynamic_int1_91467d-v15 [online], available at: http://mcu.emea.fujitsu.com/mcu_tool/detail/SK-91F467-FLEXRAY.htm [accessed 3 February 2008].

TZ Mikroelektronik (2004a) FLEXPS Dokumentation, Göppingen, Germany.

TZ Mikroelektronik (2004b) FLEXPS, Göppingen, Germany.

TZ Mikroelektronik (2007a) FlexConfig Basic, Göppingen, Germany.

TZ Mikroelektronik (2007b) FlexTiny Family Instructions for Use, Göppingen, Germany.

Vector Informatik GmbH (2007a) CANalyzer.FlexRay 7.0 Datasheet, Stuttgart, Germany.

Vector Informatik GmbH (2007b) CANalyzer 7.0 Datasheet, Stuttgart, Germany.

Vector Informatik GmbH (2007d) Hardware Interfaces for FlexRay and CAN Datasheet, Stuttgart, Germany.

Chapter 14 . Validation

14.1 Introduction

This chapter will focus on the procedure of validating the simulation model. Validation of a model is different to verification of the model. In chapter 7 it stated that verification was the process of building the model correctly while validation is concerned with determining if the right model was built. The validation procedure outlined in this chapter is designed to test if the right model was built. The procedure is determined by what the model is supposed to achieve and how it should act. The behaviour of the model was analysed by using suitable tests. The tests used will be discussed to show how the validation procedure was fulfilled. The validation test cases will then be stated and the results analysed. Any conclusions about the tests run will then be stated.

Figure 14.1 (Banks et. al. 2001, p16) shows where validation fits into the model building process.

MODEL CALIBRATION & VALIDATION

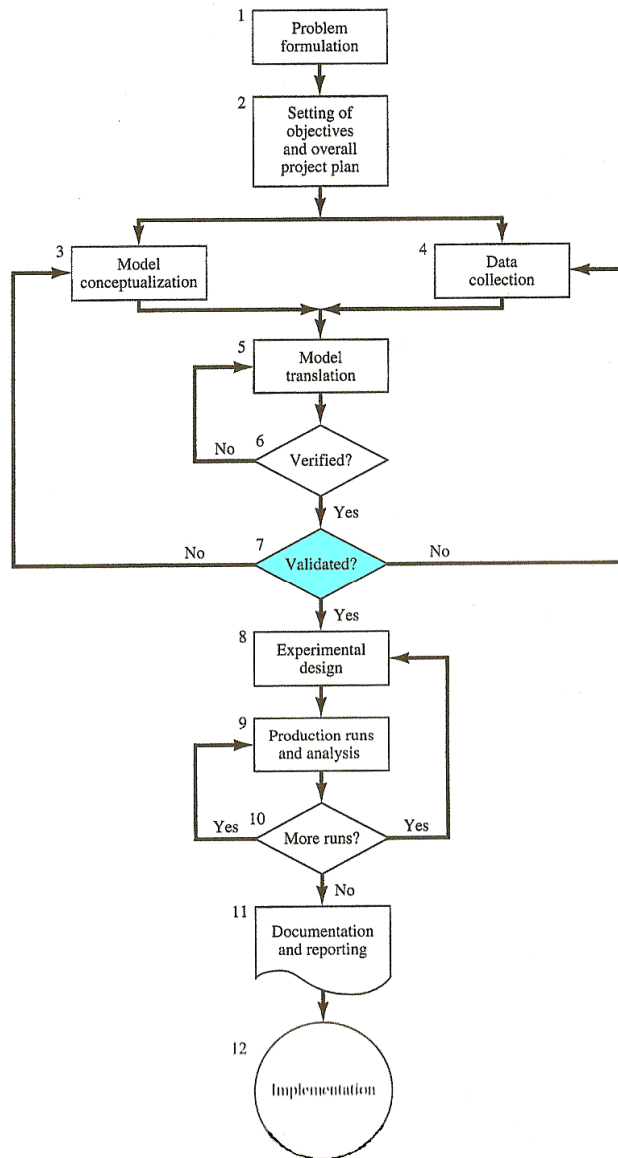


Figure 14.1: Model building process

14.2 Validation Procedure

The validation stage of a modelling study can be a difficult exercise. This is because 'no model is ever totally representative of the system under study.' (Banks et. al. 2001, p375). Each time a model is revised, as shown in Figure 14.2 (Banks et. al. 2001, p375), it increases the cost, time and effort to achieve a more accurate model. It is therefore necessary to have an idea as to what the model is intended to do and to test it for this. It is also clear that the validation stage of the model building process depends on the success of previous steps in the building process such as the verification and

MODEL CALIBRATION & VALIDATION

calibration stages. This means that if the validation stage is not a success the modeller may need to return to earlier steps in the modelling procedure.

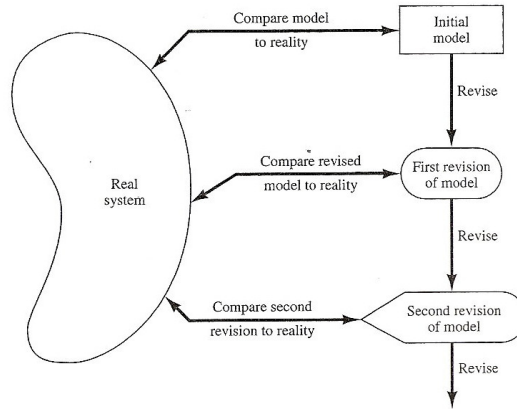


Figure 14.2: Calibration iterative process

Naylor and Finger are credited in Banks et. al. (2001, p376) as formulating a three step approach to aid the validation process. These steps are as follows:

- Build a model of high face validity.
- Validate the model assumptions.
- Compare the model input-output transformations to the real world input-output transformations.

These steps will now be discussed in relation to the study undertaken as outlined in this thesis.

- The model was demonstrated to and discussed with research supervisor Brendan Jackman during the models construction stage. Any improvements or changes were then discussed and changes implemented as necessary. The operation of the model was also tested during the verification stage of the model development. The model can therefore be said to have high face validity.
- The model was built by comparing the model to a real world system and its specifications. The behaviour of the model was also verified for accurate behaviour and calibrated against a real world system. In this way the models assumptions can be said to be validated.
- The ability of the model to make good predications of a real world system is seen as the 'ultimate test' of the model (Banks et. al. 2001, p378). To achieve

complete validation, it was necessary to conduct further tests. These tests are outlined in section 14.2.1.

14.2.1 Validation of Input-Output Transforms

The ability of a model to predict the future behaviour of a real system is the only objective test of a model. The model should be able to accurately predict the performance of the real system, if for instance the arrival rate of entities at a server is increased. When the model is able to do this the model can be considered as an accurate representation of the real world system (Banks et. al. 2001, p378).

The input-output transform validation process is then essentially a validation where accurate measures of performance are obtained from a set of given inputs. This validation procedure can be done using a separate set of historical data than that used for calibration. In this way the models behaviour can be tested in an unbiased way (Banks et.al. 2001, p378).

Section 14.3 will outline the procedure that was undertaken to validate the model according to this definition.

14.3 Validation Data Collection

The validation stage used the same technique to collect timing information as that of the calibration stage as outlined in 13.3. The same equipment and procedure that was used for calibration was used to collect real world timing data. This data was then compared to the timing data obtained from the simulation model after it was run. If the model data was deemed to be acceptably similar to the real world timing information, then the simulation model can be deemed validated. However it was also necessary to analyse usefulness of the model system.

The parameters of the validation test cases are outlined in Table 14.1. As with the calibration tests, the configuration files were made from a combination of DECOMMSYS Designer output files and FlexConfig '.chi' files.

MODEL CALIBRATION & VALIDATION

ID	Cycle Length (µs)	Number of Static Slots	Number of Mini Slots	Static Slot Length (µs)	Mini Slot Length (µs)	Static Frame Payload (words)	Dynamic Frame Payload (words max)	Channels	NIT & Symbol Length (µs)	Node Tx Frames	Node Rx Frames	Latest Tx	Note
1	5000	60	276	35	10	1	16	A&B	140	3 and 65 (A)	6 and 66 (B)	271	Based on the CANalyzer example
2	5000	79	148	39	10	8	16	A&B	439	3, and 159 (A)	6 and 155 (B)	143	Based on the BMW example
3	5000	60	245	35	10	4	20	A	450	3 and 65	6 and 66	240	Channel A only
4	9908	120	490	35	10	4	20	B	808	3 and 65	6 and 66	485	Channel B only, max NIT and double static and mini slots
5	300	6	12	27	6	2	20	A&B	66	3 and 7 (A)	6 and 8 (B)	2	Small static slot and mini slot
6	4354	60	195	39	10	8	16	A&B	64	3 and 65 (A)	6 and 66 (B)	190	Minimum NIT and Symbol window.
7	15982	2	2640	39	6	8	60	A&B	64	2 and 100 (A)	1 and 770 (B)	2617	Maximum number of mini slots and minimum static slots and NIT and Symbol window
8	5408	8	0	659	NA	127	NA	A&B	136	3	6	0	Maximum static slot length and payload

Table 14.1: Validation test case parameters

MODEL CALIBRATION & VALIDATION

The random numbers used for each test case simulation runs are shown in Table 14.2.

Test Case	Application Generation	Application Response	Physical Bus A-1	Physical Bus A-2	Physical Bus B-1	Physical Bus B-2
1	456511	578	1675424	9414884	7	2214
2	185	68132217	7506	56238	200926	205738
3	821433	303758	21417824	216	95413	445253
4	4447	660924	24963	6602275	25548	353118
5	6154	1388908	57650387	39245	72912	5971
6	79	21856	67	384193844	1829106	465256
7	9218	7426	505	60735	377	8074
8	732	938	864818	516961	632116	87399

Table 14.2: Validation test case random number seeds

14.4 Validation Review

The validation process of building a model highlights how successfully the model was built. To do this real world data was obtained for the timing and behaviour of the system for a number of test cases. The model was then set to perform the same tests and data collected. The two sets of data were then analysed for similarity. If the model produced the same information then the model was said to be validated.

14.4.1 Validation Data & Results

The data presented in this section represents the real world data that the simulation model must accurately reflect. As was stated in section 14.3, the same techniques to collect the data as used in section 13.3 were used again. The tables below, Tables 14.3 -14.10 show both the timing data for the real world system and those obtained from simulation model. This compares the actual real world timing data and the simulations models data. This is similar to the Calibration stage of comparing the real world data to the simulated data.

MODEL CALIBRATION & VALIDATION

Test Case 1		Real World Time (ns)		Simulation Model Time (ns)		Divergence (%)
Software Driver Transmit Time	Frame3	102675	Frame3	101704.9	-0.94	
	Frame65	357850	Frame65	280185	-21.70	
Software Driver Receive Time	Frame6	3791.666667	Frame6	14973.1	+294.89	
	Frame66	135093.75	Frame66	164496.1	+21.76	
Input Buffer Time	Frame3	31444.4444	Frame3	30250	-3.80	
	Frame65	31361.1111	Frame65	30250	-3.54	
Output Buffer Time	Frame6	85333.33333	Frame6	79950	-6.31	
	Frame66	85305.55556	Frame66	79950	-6.28	
Frame Transmit Time	Frame3	6100	Frame3	12800	+109.84	
	Frame65	45700	Frame65	42800	-6.35	
Frame Receive Time	Frame6	27200	Frame6	19000	-30.15	
	Frame66	54900	Frame66	49000	-10.75	

Table 14.3: Validation test case 1 data

Test Case 1 Results:

The Results for test case 1 show a similar set of results as was obtained from the calibration procedure. The majority of simulated models times fall within $\pm 10\%$ of the real world data. There are a number of data points that lie outside this range however indicating that the simulation model requires further calibration. For instance frame 6 indicates that the model is out by almost 300% at the software driver receive side while frame 3 takes almost 110% more time to transmit over the physical bus.

Test Case 2		Real World Time (ns)		Simulation Model Time (ns)		Divergence (%)
Software Driver Transmit Time	Frame3	164250	Frame3	186701.1	+27.65	
	Frame159	352550	Frame159	280185	-20.53	
Software Driver Receive Time	Frame6	61100	Frame6	84750.5	+38.71	
	Frame155	134250	Frame155	164496.1	+22.53	
Input Buffer Time	Frame3	35861.1111	Frame3	30250	-15.65	
	Frame159	35916.6667	Frame159	30250	-15.78	
Output Buffer Time	Frame6	85750	Frame6	79950	-6.76	
	Frame155	85777.8	Frame155	79950	-6.79	
Frame Transmit Time	Frame3	20111.1111	Frame3	26800	+33.26	
	Frame159	45500	Frame159	42800	-5.93	
Frame Receive Time	Frame6	31000	Frame6	33000	+6.45	
	Frame155	45000	Frame155	49000	+8.89	

Table 14.4: Validation test case 2 data

MODEL CALIBRATION & VALIDATION

Test Case 2 Results:

The data does not contain any difference between the two sets of data that is as large as some seen in test case 1. However in this test case the majority of the model data times are within a $\pm 30\%$ range of the real world system data. This again supports the need for further calibration.

Test Case 3		Real World Time (ns)		Simulation Model Time (ns)		Divergence (%)
Software Driver Transmit Time	Frame3	130725	Frame3	138497.3	+5.95	
	Frame65	386075	Frame65	325465.1	-15.70	
Software Driver Receive Time	Frame6	24916.6667	Frame6	44877.7	+80.11	
	Frame66	169944.444	Frame66	204368.9	+20.26	
Input Buffer Time	Frame3	34333.3333	Frame3	30250	-11.89	
	Frame65	34611.1111	Frame65	30250	-12.60	
Output Buffer Time	Frame6	85000	Frame6	79950	-5.94	
	Frame66	84944.4	Frame66	79950	-5.88	
Frame Transmit Time	Frame3	15000	Frame3	18800	+25.33	
	Frame65	48100	Frame65	50800	+5.61	
Frame Receive Time	Frame6	29100	Frame6	25000	-14.09	
	Frame66	54100	Frame66	57000	+5.36	

Table 14.5: Validation test case 3 data

Test Case 3 Results:

The data set for test case 3 again indicates the need for further calibration of the model. The values are within approximately $\pm 20\%$ of each other. The largest difference is seen in the software driver receive side. This is similar to the previous 3 test case results where the biggest difference was seen at this stage of the data flow.

MODEL CALIBRATION & VALIDATION

Test Case 4	Real World Time (ns)		Simulation Model Time (ns)		Divergence (%)
Software Driver	Frame3	130625	Frame3	138497.3	+6.27
Transmit Time	Frame125	386125	Frame125	325465.1	-15.71
Software Driver	Frame6	23333.3333	Frame6	44877.7	+92.33
Receive Time	Frame126	168500	Frame126	204368.9	+21.29
Input Buffer	Frame3	37638.8889	Frame3	30250	-19.63
Time	Frame125	38138.8889	Frame125	30250	-20.68
Output Buffer	Frame6	85000	Frame6	79950	-5.94
Time	Frame126	84861.1	Frame126	79950	-5.79
Frame Transmit	Frame3	14800	Frame3	18800	+27.02
Time	Frame125	48000	Frame125	50800	+5.83
Frame Receive	Frame6	29300	Frame6	25000	-14.68
Time	Frame126	92000	Frame126	57000	-38.04

Table 14.6: Validation test case 4 data

Test Case 4 Results:

The biggest difference between the two sets of data is again seen at the software driver receive side at 192%. The frame receive time is also out by about 38%. The majority of the remaining simulation data differs by approximately $\pm 20\%$.

Test Case 5	Real World Time (ns)		Simulation Model Time (ns)		Divergence (%)
Software Driver	Frame3	113900	Frame3	114030	+0.11
Transmit Time	Frame7	386125	Frame7	325465.1	-15.71
Software Driver	Frame6	6777.777778	Frame6	24941.3	+267.99
Receive Time	Frame8	171111.1111	Frame8	204368.9	+19.44
Input Buffer	Frame3	37666.6667	Frame3	30250	-19.69
Time	Frame7	38166.6667	Frame7	30250	-20.74
Output Buffer	Frame6	84888.9	Frame6	79950	-5.82
Time	Frame8	84833.3	Frame8	79950	-5.76
Frame Transmit	Frame3	7100	Frame3	14800	+108.45
Time	Frame7	48800	Frame7	50800	+4.10
Frame Receive	Frame6	18400	Frame6	21000	+14.13
Time	Frame8	63100	Frame8	57000	-9.67

Table 14.7: Validation test case 5 data

MODEL CALIBRATION & VALIDATION

Test Case 5 Results:

The difference between the two sets of data again differs in a number of stages by about $\pm 20\%$. The biggest differences are seen in the software driver receive time and frame transmit time parameters.

Test Case 6	Real World Time (ns)		Simulation Model Time (ns)		Divergence (%)
Software Driver	Frame3	164250	Frame3	186701.1	+13.67
Transmit Time	Frame65	352525	Frame65	280185	-20.52
Software Driver	Frame6	60571.42857	Frame6	84750.5	+39.92
Receive Time	Frame66	133611.1111	Frame66	164496.1	+23.12
Input Buffer	Frame3	34833.3333	Frame3	30250	-13.16
Time	Frame65	35194.4444	Frame65	30250	-14.05
Output Buffer	Frame6	85805.6	Frame6	79950	-6.82
Time	Frame66	85777.8	Frame66	79950	-6.79
Frame Transmit	Frame3	20200	Frame3	26800	+32.67
Time	Frame65	45500	Frame65	42800	-5.93
Frame Receive	Frame6	31200	Frame6	33000	+5.77
Time	Frame66	55000	Frame66	49000	-10.91

Table 14.8: Validation test case 6 data

Test Case 6 Results:

The data sets differ by approximately $\pm 20\%$ for a number of parameters. The software driver receive time and the frame transmit time are again the two values that differ by the greatest amounts. The differences at these stages are not as great as in previous stages.

MODEL CALIBRATION & VALIDATION

Test Case 7	Real World Time (ns)		Simulation Model Time (ns)		Divergence (%)
Software Driver	Frame2	169000	Frame2	186701.1	+10.47
Transmit Time	Frame100	722275	Frame100	724665.7	+0.33
Software Driver	Frame1	61541.66667	Frame1	84750.5	+37.71
Receive Time	Frame770	535138.8889	Frame770	603096.9	+12.70
Input Buffer	Frame2	37833.3333	Frame2	30250	-20.04
Time	Frame100	38611.1111	Frame100	30250	-21.65
Output Buffer	Frame1	84888.9	Frame1	79950	-5.82
Time	Frame770	85750	Frame770	79950	-6.76
Frame Transmit	Frame2	23000	Frame2	26800	+16.52
Time	Frame100	130000	Frame100	130800	-20.15
Frame Receive	Frame1	33900	Frame1	33000	-2.65
Time	Frame770	141000	Frame770	137000	-2.84

Table 14.9: Calibration test case 7 data

Test Case 7 Results:

The data sets produce a similar result to that of test case 6. The difference between the two sets is approximately $\pm 20\%$ in a number of cases. The software driver receive time and the frame transmit time are again the two values that differ by a greater amount.

Test Case 8	Real World Time (ns)		Simulation Model Time (ns)		Divergence (%)
Software Driver	Frame3	1229050	Frame3	1174996	-4.40
Transmit Time					
Software Driver	Frame6	1150500	Frame6	1270966	+10.47
Receive Time					
Input Buffer Time	Frame3	35472.22	Frame3	30250	-14.72
Output Buffer	Frame6	539138.889	Frame6	79950	-85.17
Time					
Frame Transmit	Frame3	259700	Frame3	264800	+1.96
Time					
Frame Receive	Frame6	652900	Frame6	271000	-58.49
Time					

Table 14.10: Validation test case 8 data

Test Case 8 Results:

The results for test case 8 produce the greatest difference between the two data sets. For the output buffer the two data sets correlate by only 15% approximately. For the other tests this value was never outside a $\pm 20\%$ correlation bracket. The model data for this stage of the pipeline correlated to the real world data by 93-94% in every other test case. The receive frame time also had a low correlation of about 41%. The remainder of the values correlate to about a 20% difference between the two data sets.

14.4.1.1 Validation Data Conclusion

The validation tests show a strong need to recalibrate the system. The majority of the real world and model timings observed were within a $\pm 20\%$ of each other. A number of the observed times greatly differed from the real world values and the difference was in some cases were greater than $\pm 50\%$. For the initial calibration stage a difference of about $\pm 10\%$ was desired. The differences observed at this stage are therefore not acceptable. Table 14.11 shows how many of the models subsystem timing data differed to the real world system by more than the desired $\pm 10\%$ range.

Parameters	Unacceptable Differences
Software Driver Transmit Time	7
Software Driver Receive Time	13
Input Buffer Time	12
Output Buffer Time	1
Frame Transmit Time	7
Frame Receive Time	10

Table 14.11: Validation test results summary

The building of the model is an iterative process. The model must therefore be observed for accuracy and changed as needed. This could mean a redesign of a subsystem or subsystems in the model to achieve greater accuracy. The real world data that the model was calibrated to may not be completely accurate and a new way to collect and analyse the data may be needed

14.4.2 Data Pipeline Analysis

As with the calibration analysis a measure of the transmit data and receive data pipelines of the model were done. The ability to analyse the timing of the flow of data through a FlexRay node was the ultimate goal of the simulation model. By analysing these times a measure of the accuracy of the model as a whole, and not just the subsystems can be achieved. The differences between the real world data and the simulation data is then compared the cycle time and static slot time for each test case. This gives a metric of the accuracy of the model to a known value. Table 14.12 shows the data for the transmit pipeline. Table 14.13 shows the data for the receive pipeline.

MODEL CALIBRATION & VALIDATION

Transmit Pipeline	Real World Time (ns)	Simulation Model Time (ns)	Divergence (%)	Slot Time Correlation (%)	Cycle Time Correlation (%)		
Test Case 1	Frame 3	140219.4444	Frame 3	144754.9	+3.23	12.96	0.09
	Frame 65	434911.1111	Frame 65	353235	-18.78	233.36	1.63
Test Case 2	Frame 3	220222.2222	Frame 3	243751.1	+10.68	60.33	0.47
	Frame 159	433966.6667	Frame 159	353235	-18.60	207.00	1.61
Test Case 3	Frame 3	180058.3333	Frame 3	187547.3	+4.16	21.40	0.15
	Frame 65	468786.1111	Frame 65	406515.1	-13.28	177.92	1.25
Test Case 4	Frame 3	183063.8889	Frame 3	187547.3	+2.45	12.81	0.05
	Frame 125	472263.8889	Frame 125	40651.1	-13.92	1233.18	4.36
Test Case 5	Frame 3	158666.6667	Frame 3	159080	+0.26	0.15	0.14
	Frame 7	473091.6667	Frame 7	406515.1	-14.07	24.66	22.19
Test Case 6	Frame 3	219283.3333	Frame 3	243751.1	+11.16	62.74	0.56
	Frame 65	433219.4444	Frame 65	353235	-18.46	205.09	1.84
Test Case 7	Frame 2	229833.3333	Frame 2	347751.1	+51.31	302.35	0.74
	Frame 100	790886.1111	Frame 100	885715.7	+11.99	243.1528	0.59
Test Case 8	Frame 3	1524222.22	Frame 3	1470046	-3.55	8.220974	1.00
Average Correlation					-0.36	187.02	2.44

Table 14.12: Transmit pipeline timing

The accuracy of the transmit model pipeline falls well within a communication cycle length, with the greatest difference at 22.19% of a communication cycle. The accuracy compared to a static slot varies greatly from 0.15% of a static slot time to

MODEL CALIBRATION & VALIDATION

302%. The majority of the values that have an accuracy greater than one static slot are for dynamic messages.

Receive Pipeline	Real World Time (ns)	Simulation Model Time (ns)	Divergence (%)	Slot Time Correlation (%)	Cycle Time Correlation (%)		
Test Case 1	Frame 6	116325	Frame 6	113923.1	-2.06	6.86	0.05
	Frame 66	247599.3056	Frame 66	293446.1	+18.52	130.99	0.92
Test Case 2	Frame 6	177850	Frame 6	197700.5	+11.33	50.90	0.40
	Frame 155	265027.8	Frame 155	293446.1	+10.72	72.87	0.57
Test Case 3	Frame 6	139016.6667	Frame 6	149827.7	+7.78	30.89	0.22
	Frame 66	308988.844	Frame 66	341318.9	+10.46	92.37	0.65
Test Case 4	Frame 6	137633.3333	Frame 6	149827.7	+8.86	34.84	0.12
	Frame 126	345361.1	Frame 126	341318.9	-1.17	11.55	0.04
Test Case 5	Frame 6	110066.6778	Frame 6	125891.3	+14.38	5.86	5.27
	Frame 8	319044.4111	Frame 8	341318.9	+6.98	8.25	7.42
Test Case 6	Frame 6	177577.0286	Frame 6	197700.5	+11.33	51.60	0.46
	Frame 66	274388.9111	Frame 66	293446.1	+6.95	48.86	0.44
Test Case 7	Frame 1	180330.5667	Frame 1	197700.5	+9.63	44.54	0.11
	Frame 770	761888.8889	Frame 770	820046.9	+7.63	149.12	0.364
Test Case 8	Frame 6	2342538.889	Frame 6	1621916	-30.76	109.35	13.33
Average Correlation					+6.04	56.59	2.02

Table 14.13: Receive pipeline timing

MODEL CALIBRATION & VALIDATION

The accuracy of the receive pipeline model pipeline falls well within a communication cycle length, with the greatest difference at 13.33% of a communication cycle. The accuracy compared to a static slot varies greatly from 5.86% of a static slot time to 149%. The majority of the values that have an accuracy greater than one static slot are again for dynamic messages.

As the accuracy of the pipelines is greater than some of the individual stages of model, the allocation of timing to the individual subsystems may be incorrectly allocated. One subsystem takes longer compared to the real world system, while a following subsystem takes a shorter time. The various timings of the subsystems could therefore be adjusted to more accurately reflect the real world systems.

As the dynamic messages are generally less accurate than static messages, the model may need to be studied from this perspective to achieve greater accuracy.

14.4.3 Buffer Access Time Analysis

The model must be able to also produce a set of results that are useful to a systems analyst. The purpose of the model was to produce an output that could be used to improve the overall system. By analysing the timing of the individual model subsystems, bottlenecks could be detected and the system improved. In a number of cases the delays of the data may not be decreased in a simple manner. A system analyst may then have to find another way to increase efficiency in the system. By analysing the time a buffer is accessed (this could be to read from or write to the buffer), an improvement may be made in the design of the application layer's execution.

Table 14.1 shows the time a buffer is checked by the E-Ray chip for transmission (Request to Send Time) and this is compared to the time the buffer is updated by the host (Buffer Update Time).

MODEL CALIBRATION & VALIDATION

Buffer Update	Frame ID	Request to Send Time (ns)	Buffer Update Time (ns)	Difference (ns)
Test Case 1	3	35000	132454.9	-97454.9
	65	2140000	443389.9	1696610.0
Test Case 2	3	39000	217451.1	-178451.1
	159	3821000	528386.1	3292613.9
Test Case 3	3	35000	169247.3	-134247.3
	65	2140000	525462.4	1614537.6
Test Case 4	3	35000	169247.3	-13427.3
	125	4230000	525462.4	42704537.6
Test Case 5	3	27000	144780	-117780
	7	135000	439300	-439300
Test Case 6	3	39000	217451.1	-178451.1
	65	2370000	528386.1	1841613.9
Test Case 7	2	39000	217451.1	-178451.1
	100	654000	972866.8	-318866.8
Test Case 8	3	659000	1205746	-546746

Table 14.14: Buffer update time

In Table 14.14 it can be clearly seen that there is a big difference between the time a buffer is updated by the application and the time the buffer is checked for transmission. A negative value indicates that a buffer was updated after the E-Ray checked the buffer, while a positive value indicates the buffer was updated before this time. All the times shown for the E-Ray chip to check a buffer assigned to dynamic frames is relative to a best case scenario for the frame to be transmitted.

Table 14.15 shows the difference between when a buffer is updated from a frame transmitted over the physical bus and stored in a buffer (Buffer Update Time) to the time the software driver attempts to read the buffer (Request to Read Time). As with the data in Table 14.14, there can be a large time difference when a message may wait in a buffer (this is indicated by a positive difference in the Table 14.15). The negative differences correspond a time where a buffer was accessed before the buffer was updated during the current communication cycle.

MODEL CALIBRATION & VALIDATION

Buffer Update	Frame ID	Request to Read Time (ns)	Buffer Update Time (ns)	Difference (ns)
Test Case 1	6	448189.9	194000	254189.9
	66	543613	2199000	-1655387
Test Case 2	6	528386.1	228000	300386.1
	155	698386.6	3865300	-3166913.4
Test Case 3	6	530262.4	200000	330262.4
	66	655590.1	2257000	-1601409.9
Test Case 4	6	530262.4	200000	330262.4
	126	655590.1	4357000	-3701409.9
Test Case 5	6	225000	156000	69000
	8	611186.3608	225000	386186.3608
Test Case 6	6	533186.064	228000	305186.064
	66	698386.6	2439000	-1740613.4
Test Case 7	1	977666.8128	33000	58166.8128
	770	1142867	4817000	-3674133
Test Case 8	6	1210546	3566000	-2355454

Table 14.15: Buffer read time

By analysing this data it may be possible to improve the flow of data around the system. For instance an analyst may discover that a message assigned to the third slot can never be updated at the start of the current communication slot. It may be necessary to then update the buffer at the end of a communication cycle to reduce the latency experienced by a message before it transmitted. It may also be seen that a receive message buffer should be read at a later time in the communication cycle to obtain the most up-to-date data.

14.4.3.1 Buffer Utilisation Analysis

At the start of the model building process it was hoped that the buffer usage of the system could be analysed. As static messages are generally time-critical messages they should have assigned buffers that will always available to store data to or read data from. Therefore the access time analysis (as in section 14.4.3) of a buffer is the most efficient way to analyse the buffering 'utilisation'. This also applies to dynamic messages with assigned buffers. As these buffers will always be available to store data

to or read data from, the access times of the buffers should be analysed to optimise the system.

The FIFO is a dynamic system that can store a number of messages. In general static messages will not be stored in the FIFO and so only dynamic messages should be stored in the FIFO. However the FIFO rejection filters will store all messages with a frame ID within a given range. This means that the FIFO could store data not utilised by the node's application. Therefore by analysing the utilisation of the FIFO, messages may be assigned to a dedicated receive buffer to help achieve an optimal configuration.

No timing data for the FIFO could be obtained and as such no tests could be run. However this could become an important aspect in both the real world and the simulation model systems.

14.4.4 Application Layer Execution Time Analysis

The model output was again analysed for useful information. It was seen that length of time a message spends in the software driver stage was a significant part of the overall data flow. The application layer must wait for the software driver to complete its operation before more it can execute another instruction. As the application layer should be synchronised to the communication cycle the amount of time the application can execute for is limited before it must restart the task. If it is known how much time is allotted to the software driver, the remaining time can then be assigned to other operations such as data processing. Table 14.16 shows the percentage of the communication cycle that the application layer has to wait for the software driver to perform all its tasks.

MODEL CALIBRATION & VALIDATION

Test Case	Total Software Driver Time (Real World) (ns)	Total Software Driver Time (Model) (ns)	Cycle Length Percentage (%)	
			Real World	Model
1	822325	721259.1	16.45	14.43
2	946925	876032.7	18.4	17.52
3	947000	873109	18.94	17.46
4	946925	873109	9.56	8.81
5	863100	828705.2608	287.70	276.24
6	737525	876032.664	16.94	20.12
7	1735000	1759114.213	10.86	11.01
8	2426175	2525912.631	44.86	46.71

Table 14.16: Total software driver times

As can be seen from Table 14.16 a large amount of time may be spent by the application layer waiting for the software drive to complete all the tasks. In test case 5 the percentage is almost 300%, indicating that the application layer could not run in one communication cycle. This could mean that the task may need to be split over a number of nodes. Likewise for test case 8, there is almost half the execution time is taken over by the software drive passing information between the host MCU and the E-Ray communications controller. The remaining time may be insufficient to process all the data during one communication cycle.

14.4.5 Validation Data with Random Number Generator

The function of the physical bus layer was to generate a set of frames that simulate the traffic on the FlexRay network. The calibration and validation stages, as described in chapters 13 and 14, required that the physical bus subsystem of the model to be set up to mimic the two node real world system that was implemented in W.I.T. by removing the random nature of the physical bus subsystems frame generation. This meant that the operation of the physical bus could not be fully validated over these tests.

To achieve a validation of the physical bus layer it was necessary to implement the same validation test cases with the random element reintroduced. The physical bus subsystem data was analysed and then compared to the desired outcome. For each test case the physical bus generated a random set of frames from the physical bus that were either accepted or rejected by the node. This was the desired operation of the physical

bus subsystem. This would mean a systems analyst could apply a FlexRay network frame pattern to the physical bus subsystem and a reasonable sent of frames would then be produced by the simulation model. This ultimately would allow an analyst achieve a good insight into the optimal timing for dynamic frames buffer access.

14.5 Conclusion

By following the steps as outlined in this chapter, chapter 14, the model was tested against a real world system. To ensure the model was an accurate representation of the system under investigation, a real world data set was obtained from the real world system. The model was run with the new test case constraints also applied. The timing of the system was then checked against the real world system data. A number of frames were not passed to the physical bus by the simulation model communications controller. Further analysis of the data showed that this was due to message buffers not being updated in time for the frames assigned to the static segment. The dynamic frames that were not transmitted were not transmitted as the dynamic slot was never reached during the simulation run. Likewise for the frames received from the FlexRay bus model, some frames were never stored. This was due to the physical bus not generating a frame during the given slot. For each of the tests the behaviour of the simulation model was therefore validated.

‘Validation is concerned with building the right model’ (Banks et. al. 2001, p367). This is the ultimate test for the simulation model. The simulation model that was built and tested as described in this thesis was to be used to analyse the flow of data through a FlexRay based system. To be able to analyse the flow of data a number of subsystems were implemented to logging timing of various entities as they pass through the various model subsystems. In section 14.4.1.2 the buffer access times were analysed based on the data obtained from the simulation model. From this the data showed that a number of deadlines were missed to update buffers. This meant that a message would have to wait for the next communication cycle to transmit the data. Also the application driver may attempt to read data before new data is stored in the buffer. By analysing the model data time constraints for the application layer can be calculated. By modifying the application layer model to match the timing constraints the new configuration can be tested. In section 14.4.1.3 the software driver execution time was analysed. This

MODEL CALIBRATION & VALIDATION

produced a metric that describes the total driver execution time over a communication cycle length. This is the minimum time the application layer will execute an instruction for. If this is known the remaining time can be assigned to calculations or system checks. An optimum application configuration could then be developed and tested using the model. The ability to extract and analyse this data therefore meets the purpose of the simulation model. The right model has therefore been built.

From the behaviour of the simulation model and the model's outputs there is a strong suggestion that the correct model was built. However this stage of the model development cycle has indicated a strong need for further calibration and validation of the timing constraints of the model.

Figure 14.3 (Banks et. al. 2001, p16) displays the stages after the model is fully validated and calibrated as working correctly to a desired level of accuracy. Only after a simulation model has reached an acceptable level of accuracy can confidence be placed in the simulation model to accurately reflect the real world system. From the highlighted segments of Figure 14.3 it can be seen that experiments can then be conducted using the model. The results of the system can then be analysed and improvements for the simulation system noted. The simulation can then be run with these implemented. If the test results produce the desired output then the real system can then be implemented with these improvements incorporated.

MODEL CALIBRATION & VALIDATION

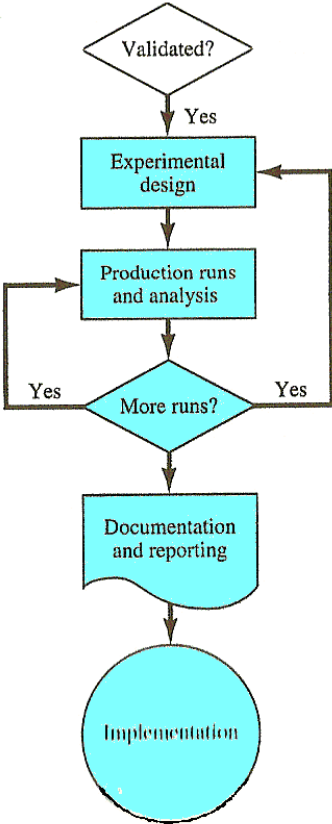


Figure 14.3: Final model steps

14.6 References

Banks, J., Carson, J. S., Nelson, B. L. and Nicol, D. M. (2001) Discrete-Event System Simulation, New Jersey: Prentice Hall.

Section V:
Conclusion

Chapter 15 . Conclusion

15.1 Introduction

The aim of the research outlined in this thesis was to develop a method to optimise a FlexRay node. This is to ease the transition to this new communication protocol. The use of FlexRay could lead to an increase in efficiency and reliability of the communication network in automobiles. In order to achieve the optimisation, it was necessary to look at industry standard tools as well as the FlexRay protocol.

15.2 Research Summary

To achieve the optimisation of a FlexRay node a simulation model was developed. This consisted of the application, software driver, communication controller and the FlexRay physical bus layers. The individual model subsystems were based on industry standard tools and systems such as the Bosch E-Ray communications controller.

The communications controller was modeled on the Bosch E-Ray controller and the software driver was based on the DECOMSYS FlexRay driver 'COMMSTACK'. The application layer was based on sample software provided for the Fujitsu SK-91F467-FlexRay development boards. It was also necessary to develop a communications bus over which data could be sent and received. This layer needed to generate a number of messages that the simulated node may accept. The physical layer model must also accept messages from the simulated node. This model was then tested for its suitability to perform the desired functionality.

CONCLUSION

15.3 Research Questions

At the start of the research a number of questions were asked. These research questions will again be listed and then answers proposed.

15.3.1 Research Questions

1. What aspects of the FlexRay configuration most affects the performance and design of distributed vehicle applications?
2. What guidelines should be used to configure the protocol stack for best application performance?
3. What techniques can be used to optimise local buffer usage for specific vehicle applications using a fixed global network message schedule?

15.3.2 Research Answers

1. FlexRay is a complicated protocol with many variables and constraints. The main benefit of the FlexRay protocol is the combination of both time-triggered and event-triggered segments in the same system. The event-triggered segment allows the designer of a system to assign messages to different priorities for non critical systems. The time-triggered segment allows a system to be implemented with guaranteed known message latency. In order to maximise the effectiveness of the protocol there are a number of areas that should be looked at. These areas include the communication cycle and the distribution of messages to either static or dynamic messages. The allocation of frames to static or dynamic messages will have a big impact on the arrival of messages to nodes on the bus. From this research it was also found that the application layer could also have a big impact on the system. If the application cannot process frames arriving at a node in a timely fashion, then errors may be created. These errors include calculations based on old information or a message buffer being updated after it is checked for transmission. A system designer should there analyse the flow of data from the application layer. The optimal execution order of application layer tasks will ensure deadlines are met throughout the entire node. All aspects need to be considered and the longest delays of messages accounted for.

CONCLUSION

2. The protocol stack is a combination of the physical bus over which information is transmitted, the communications controller implementation, software driver and the application. To ensure that the entire stack performs correctly none of these aspects can be ignored. When configuring the protocol stack the following will apply:

- An application, running on a processor with insufficient resources, may not execute in the desired manner. The application will be affected by various different factors also and this must be accounted for. For instance, the longest time seen in the flow of data around a node, during the research, was that of the software driver execution delay. A system designer must allow sufficient time to execute all tasks including the software delay. In chapter 3 current FlexRay products currently on the market were discussed. This included software modules to implement FlexRay systems. If these are used it may be difficult to understand the execution of the software and this could mean it may be difficult to meet deadlines in the system. Analysis of the other system stack parameters may therefore lead to the best configuration.
- The application will also be effected by the various factors relating to the communication cycle. This was seen in the validation chapter where the test case 5 application layer had insufficient time to execute in a single communication cycle. It is up to the developer of the application to fit to the predefined cycle length.
- It is necessary for the system designer to ensure the message buffers are read before new information is received and stored over any unread data. Likewise the application should be respond to messages in the required time. Again from the validation chapter it was seen that a message may reside in a receive buffer for a long period before an attempt to read the data is performed. Optimising the message buffer access will ensure efficient communication is achieved.
- Using cycle multiplexing techniques allows different information to be sent during different communication cycles for a given slot. This could be taken into account when designing the application and allow

CONCLUSION

messages of different generation periods to implemented on the same node and network.

- The number of FlexRay networks implemented in a vehicle as well as the number of nodes used should be investigated. By splitting the nodes into different functional networks, an optimisation of the automotive applications could be achieved. This could mean for example that all body domain information could be sent over a separate network to any engine or comfort system networks. The different networks could then be connected if necessary using a number of ‘gateway’ nodes.
3. A simulation model of a communications system can be used to gain an insight into how a system performs. The model could be setup to display a number of different statistics. These could include the utilisation of a message buffer. For instance, message ID_n could be assigned a dedicated receive buffer in the message RAM of a communications system. By observing the buffer usage it could be discovered that this buffer is rarely updated. The analyst could then decide that the message buffer could be assigned to message ID_m and message ID_n could be stored in the FIFO.

The simulation model could also indicate that a message buffer is never read by the application. This could mean that the application takes too long to execute and can never reach a point when the application requests the message buffers contents. This could be an indicator that the application needs to be moved to another node with more resources. This will lead to an optimal configuration can be achieved for the overall global communications system.

15.4 Research Conclusions

A way to analyse the flow of data through a FlexRay node has been developed and presented in this thesis. By analysing the data flow thoroughly it is possible that huge benefits could be obtained. The deadlines for messages can be analysed and any potential bottlenecks accounted for. The flow of data can then be guaranteed from the application layer down to actual transmission time.

CONCLUSION

Another big advantage of this system is the ability to analyse the buffer usage. It may be found that the buffer allocation is insufficient or unnecessary. This could allow messages to be swapped between a dedicated receive buffer and the FIFO to optimise the system. The following extract from an e-mail received from the Elektrobit support-team (Skorepa 2008) highlights the use of the FIFO in industry.

“FIFO Support is currently no official feature of Designer Pro. FIFO support exists in the engineering [sic] version of Designer Pro for BMW. A minimal [sic] FIFO support for the full version of Designer Pro is on the feature list for 2009.”

From the e-mail it can be seen that there is a desire by industry to use FIFO message buffers in FlexRay applications. Only proper analysis of this will therefore produce optimal configurations.

In conclusion a simulation of the communications controller, used in a FlexRay based system, could help increase optimisation and reduce development time and costs.

15.4.1 Observations

A number of observations have been made about the simulation model and its testing procedure. These are as follows:

- The testing was split up into two different sections, verification and validation testing. This ensured that the model and its individual subsystems performed as intended. This testing procedure also meant that no biased conclusions would be drawn from the validation section of the testing.
- The testing was designed to see if the model would accurately reflect a real world system. These tests were therefore designed to include a wide range of parameters including the maximum and minimum constraints that any system may observe in a real-world system. The tests included two real-world examples thus ensuring that realistic system constraints were included during the testing.
- It was noticed during testing that the model would never transmit during slot 1 of any communication cycle. However there was insufficient time to fix this problem. It was noted that if anyone desired to test a system where the node was to setup to transmit during slot 1 this could be worked around. The proposed

CONCLUSION

solution is that the tester simply needs to increase the frame IDs by one. This should have minimal effect on the buffering or timing analysis of the communications controller.

- SimEvents is a powerful tool. It is part of the MATLAB software program. This provides SimEvents with powerful data analysis tools. This is a huge advantage as data does not need to be exported to another program for analysis. However the version used to develop the simulation model has a few bugs and problems. Based on correspondence with Michael Clune (Clune 2008) it was discovered that some problems associated with developing the message RAM was a well known problem.

“While you have made good progress in creating, what amounts to admission control or a semaphore circuit that controls the number of entities that can access a region of the model, the access is presently limited to one 'in' path and one 'out' path and the logic is fairly easy to track. The complexity will, most likely come when there are multiple input entity paths and multiple output entity paths. With this, the calculation complexity grows quite quickly. Once we recognized that, we realized that we needed a pair of blocks to make the admission control more scalable and easier to model. This is the motivation for the Entity Combiner and Entity Splitter blocks. The current version of SimEvents contains these blocks and some demos for using them effectively. Also, I am working on a demo that models multiple CPUs accessing a common memory chip with access control modeled [sic] by these blocks.”

The newest SimEvents version provides blocks that were mentioned in the e-mail from Michael Clune. These blocks were developed to cut down on the processing time. The blocks were also specifically designed to perform the message handler access type of problem. The improvements would be a great benefit to any other simulation model that was developed.

- The method that SimEvents uses to ensure the correct interleaving of blocks is to use a single server (of zero service time) in parallel with a discrete event subsystem block which would perform a calculation in zero simulation time. The single server block creates a block where an entity can reside. When modeling buffers this is not always desirable as this method may create, if the developer is not careful, what could be seen as an extra buffer. This is especially

CONCLUSION

important in the case where the number of buffers is small i.e. one or two buffers. Due care must therefore be taken to ensure that the system performs in the correct manner.

- The time to setup all the model parameters can be time consuming. Also when data is obtained from the model there can be a lot of information to analyse. To improve the usefulness of the model a front end application could be developed. This wasn't done due to time constraints. However a front end application could be used to import either a '.chi' file or the output configuration file from the Designer Pro software. These files would contain most of the necessary information to setup the model for execution. The front end application would automate the setup process, thus greatly reducing the setup time, while also eliminating any potential mistakes. The front end application could be used to display the information obtainable from the simulation model in a user friendly way. These two additions would greatly increase the effectiveness of the system.

15.5 Area of Further Study

There are a number of possible research opportunities in the area of FlexRay. This is due to the relatively young age of the protocol. This is highlighted by the fact that the FlexRay consortium has extended the consortium agreement past its initial expiration date for an extra year. The new agreement is due to expire on the 31st December 2009 (FlexRay Consortium 2008). Possible research topics include:

- The application layer and its implementation. There are no set guidelines on how the application should be implemented. In many instances that the author has come across the application has been based on synchronizing to the communications cycle. However there is no research that looks at implementing the application as an event-triggered system based on interrupts. The effects of implementing such a mixed event-triggered/time-triggered system are therefore unknown for FlexRay systems. There is a possibility that a design methodology could be developed to implement and optimise a wide variety of different application types.

CONCLUSION

- When an event-triggered application is compared to a system where the application is more closely synchronized to the communication cycle observations would be made about the accuracy of each system. In turn this could lead to an investigation into the hardware needed to achieve an optimal system configuration. A separate standalone communications controller cannot convey to the same accuracy, the current communication time of the system than that of a system where the host and communications controller are integrated into a single electronic chip. There is already a number of microcontrollers available with built in FlexRay features (Fujitsu Microelectronics Europe 2006) and this further highlights the need to conduct such a study.
- There are a number of implementations of FlexRay drivers. Some are basic software drivers, while others provide increased functionality. There are also a number of AUTOSAR FlexRay stacks. The effect on the timing of a system could be greatly affected by the software driver implementation. It is therefore possible that research could be conducted into the optimisation of this aspect of FlexRay.

15.6 References

Berwanger, J., Schedl, A. and Peller, M (2004) BMW – First Series Cars with FlexRay in 2006, Automotive electronics + systems Special Edition, Development Solutions 19 for FlexRay ECUs, 6-8.

Clune, M. (2008) RE: Denying access to entities until a task is finished in SimEvents, email to Robert Shaw (rshaw@wit.ie), 8 December [accessed 15 Dec 2008].

Fujitsu Microelectronics Europe (2006) MB91F465XA Factsheet, Langen: Fujitsu Microelectronics Europe GmbH.

CONCLUSION

Skorepa, P. (2008) Subject: I am attempting to set up a FIFO using Designer PRO <Light> {HD - TicketID 2220LJL}, email to Robert Shaw (rshaw@wit.ie), 16 December [accessed 16 Dec 2008].

FlexRay Consortium (2008) About FlexRay [online], available: <http://flexray.com/> [accessed 9 December 2008].

Section VI:
Bibliography

BIBLIOGRAPHY

Ademaj, A., Sivencrona, H., Bauer, G. and Torin, J. (2003) Evaluation of Fault Handling of the Time-Triggered Architecture with Bus and Star Topology, Proceedings of the 2003 International Conference on Dependable Systems and Networks, San Francisco, California, June 22 - 25, 2003, IEEE Computer Society Washington, DC, USA, 123-132.

Aidemark, J., Folkesson, P. and Karlsson, J. (2005) A Framework for Node-Level Fault Tolerance in Distributed Real-Time Systems, Proceedings of the 2005 International Conference on Dependable Systems and Networks, Yokohama, Japan, June 28 - July 1, IEEE Computer Society Washington, DC, USA, 656 – 665.

Airtricity® Holdings Limited (2007) Kyoto Protocol [online], available: http://www.airtricity.com/ireland/environment/kyoto_protocol/ [accessed 19 November 2007].

Altium Limited (2006) Connecting Memory and Peripheral Devices to a 32-bit Processor v1.1, New South Wales, Australia.

Amar, J. G. (2006) The Monte Carlo Method in Science and Engineering, Computing in Science and Engineering, 8(2), 9-19.

AUTOSAR GbR (2008) Specification of the FlexRay Driver version 2.2.1, Munich, Germany.

AUTOSAR GbR (2008) Specification of the FlexRay Interface version 3.0.2, Munich, Germany.

AUTOSAR GbR (2008) Specification of the FlexRay Network Management version 3.0.2, Munich, Germany.

AUTOSAR GbR (2008) Specification of the FlexRay Transceiver Driver version 1.2.2, Munich, Germany.

BIBLIOGRAPHY

AUTOSAR GbR (2008) Specification of the FlexRay Transport Layer version 2.2.1, Munich, Germany.

Awan, I., Yar, A. and Woodward, M.E. (2006) Analysis of Queueing Networks with Blocking under Active Queue Management Scheme, Proceedings of the 12th International Conference on Parallel and Distributed Systems, Minneapolis, Minnesota, USA, July 12-15 2006, IEEE Computer Society Washington, DC, USA, 61-68.

Barr, M. (1999) Programming Embedded Systems, California: O'Reilly & Associates, Inc.

Bauer, D., Yaun, G., Carothers, C., Yuksel, M. and Kalyanaraman, M. (2004) A Case Study in Meta-Simulation Design and Performance Analysis for Large-Scale Networks, Ingalls, R.G., Rossetti, M.D., Smith, J.S. and Peters, B.A., eds., Proceedings of the 2004 Winter Simulation Conference, Washington D.C., USA, December 5-8 2004, , IEEE Computer Society Washington, DC, USA, 206 – 214.

Beichl, I. and Sullivan, F. (2006) Guest Editors' Introduction: Monte Carlo methods, Computing in Science and Engineering, 8(2), 7-8.

Berwanger, J., Schedl, A. and Peller, M (2004) BMW- First Series Cars with FlexRay in 2006, Automotive electronics + systems, Development Solutions 19 for FlexRay ECUs, 6-8.

BMW Manufacturing Co. (2006) THE NEW BMW X5
Perfect Blend of Driving Dynamics, Functionality and Exclusivity [press release], 8 August, available:
http://www.bmwusfactory.com/media_center/releases/release.asp?intReleaseNum=209&strYear=2006 [accessed 2 October 2007].

Böhm, S. (2005) H-RAFT – Heuristic Reachability Analysis for Fault Tolerance Protocols Modelled in SDL, Proceedings of the 2005 International Conference on

BIBLIOGRAPHY

- Dependable Systems and Networks, Yokohama, Japan, June 28 - July 1, IEEE Computer Society Washington, DC, USA, 466 - 475.
- Brey, B. (1998) *Embedded Controllers: 80186, 80188, and 80386EX*, New Jersey: Prentice Hall.
- Buchana, W. (1996) *Applied PC Interfacing, Graphics and Interrupts*, Harlow: Addison Wesley.
- Carley L. (2006) *Controller Area Network (CAN) Diagnostics* [online], available: http://www.aalcar.com/library/can_systems.htm [accessed 12 June 2008].
- Carlson, J., Lennvall, T. and Fohler, F. (2003) *Enhancing Time Triggered Scheduling with Value Based Overload Handling and Task Migration*, in *Proceedings of the Sixth IEEE International Symposium on Object Oriented Real-Time Distributed Computing*, Hokkaido, Japan, May 14-16, 2003, IEEE Computer Society Washington, DC, USA, 121-128.
- Chadwick, J. (2006) *A Timing Analysis of Automotive Time-Triggered Systems*, unpublished thesis (M.Sc.), Waterford Institute of Technology.
- Chung, L. Ma, W. and Cooper, K. (2006) *Requirements Elicitation through Model-Driven Evaluation of Software Components*, *Proceedings of the fifth International Conference on Commercial-off-the-Shelf (Cots)-Based Software Systems*, Bilbao, Spain, February 7-11 2005, IEEE Computer Society Washington, DC, USA, 187-196.
- Claesson, V., Ekelin, C. and Suri, N. (2003) *The Even-Triggered and Time-Triggered Medium-Access Methods*, *Proceedings of the Sixth IEEE International Symposium on Object-Oriented Real-Time Distributed Computing*, Hakodate, Hokkaido, Japan, May 14-16, 2003, IEEE Computer Society Washington, DC, USA, 1-4.
- Corbet, J., Rubini, A. and Kroah-Hartman, G. (2005) *Linux Device Drivers*, Third Edition, California: O'Reilly & Associates, Inc.

BIBLIOGRAPHY

de Mesquita, M.A. and Hernandez, A.E. (2006) Discrete-Event Simulation of Queues with Spreadsheets: A Teaching Case, Perrone, L.F., Wieland, P.F., Lui, J., Lawson, B.G., Nicol, D.M. and Fujimoto, eds., Proceedings of the 2006 Winter Simulation Conference, Monterey, California, USA, December 3-6, 2006, IEEE Computer Society Washington, DC, USA, 2277 – 2283.

Demmeler, T. and Giusto, P. (2001) A Universal Communication Model for an Automotive System Integration Platform, Proceedings of Design, Automation, and Test in Europe, Munich, Germany, 13-16 March 2001, IEEE Computer Society Washington, DC, USA, 47-54.

Denbigh, P. (1998) System Analysis & Signal Processing, Harlow: Addison Wesley.

Dennis, A., Wixom, B.H. and Roth, R.M. (2006) Systems Analysis & Design, New Jersey: Wiley.

Dependable Computer Systems (2007) DESIGNER PRO 4.3.0 - DESIGNER PRO, DESIGNER PRO <LIGHT> and DESIGNER PRO <SYSTEM> Document Version 2.2, Wein, Austria.

Dependable Computer Systems (2007) DESIGNER PRO 4.3.0 - DESIGNER PRO, DESIGNER PRO <LIGHT> and DESIGNER PRO <SYSTEM> Document Version 2.2, Vienna, Austria.

Di Natale, M. (2006) Optimizing the Multitask Implementation of Multirate Simulink Models, Proceedings of the Twelfth Real-Time and Embedded Technology and Applications Symposium, San Jose, California, United States, April 4-7 2006, IEEE Computer Society Washington, DC, USA, 335 - 346

Ding, S., Murakami, N., Tomiyama, H. and Takada, H. (2005) A GA-Based Scheduling Method for FlexRay Systems, Proceedings of the International Conference on Embedded Software, Jersey City, New Jersey, USA, September 19-22 2005, IEEE Computer Society Washington, DC, 110 – 113.

BIBLIOGRAPHY

Elmenreich, W., Bauer, G. and Kopetz, H. (2003) The Time-Triggered Paradigm, Proceedings of the Workshop on Time-Triggered and Real-Time Communication, Manno, Switzerland, December 12.

Epstein, M., Tiwari, A., Shi, L. and Murray, R. (2005) Estimation of Linear Stochastic Systems Over a Queueing Network, Proceedings of the 2005 Systems Communications, Montreal, Canada, August 14-17 2005, IEEE Computer Society Washington, DC, USA, 389-394.

Field, T., Harder, U. and Harrison, P. (2002) Network Traffic Behaviour in Switched Ethernet Systems, Proceedings of the 10th IEEE International Symposium on Modelling, Analysis and Simulation of Computer and Telecommunications Systems, Fort Worth, Texas, USA, October 11-16 2002, IEEE Computer Society Washington, DC, USA, 243 – 260.

Freescale Semiconductor (2008) Local Interconnect Network [online], available: <http://www.freescale.com/webapp/sps/site/application.jsp?nodeId=022050123D> [accessed 12 June 2008].

FrontRunner Computer Performance Consulting (2002) Eight Steps to a Successful Performance Study [online], available: <http://www.frontrunnerepc.com/> [accessed 28 May 2008].

Fujitsu (2008) Applications – Body and Comfort Electronics [online], available: <http://www.fujitsu.com/emea/services/industries/automotive/microelectronics/applications.html> [accessed 12 June 2008].

Fujitsu Limited (2007) FlexRay ASSP MB88121B User's Manual, Japan.

Fujitsu Microelectronics Europe (2005) MB88121 Factsheet, Langen: Fujitsu Microelectronics Europe GmbH.

BIBLIOGRAPHY

Fujitsu Microelectronics Europe (2006) FlexRay 32-bit Microcontroller FlexRay-FPGA-EVA-KIT-369, Hardware Modification for Interrupt Usage Application Note, Revision 1.0, Langen: Fujitsu Microelectronics Europe GmbH.

Fujitsu Microelectronics Europe (2006) FlexRay 32-bit Microcontroller FlexRay-FPGA-EVA-KIT-369, Interrupt Usage On FlexRay CC Application Note, Revision 1.0, Langen: Fujitsu Microelectronics Europe GmbH.

Fujitsu Microelectronics Europe (2006) MB91F467D Preliminary Datasheet, MB91460 Series, Revision 0.995, Langen: Fujitsu Microelectronics Europe GmbH.

Fujitsu Microelectronics Europe (2007) Fr Family 32-bit Microcontroller

Fujitsu Microelectronics Europe (2008) FlexRay 32-bit Microcontroller SK-91F467-FlexRay, Using SPI on SK-91F467-FlexRay, Application Note, Langen, Germany.

Fujitsu Microelectronics Europe GmbH (2005) Automotive Solutions CMOS FlexRay ASSP MB88121/MB88121A/MB88121B/MB88121C preliminary datasheet, Revision 1.0, Langen: Fujitsu Microelectronics Europe GmbH

Garmus, D. and Herron (1996) Measuring the Software Process- a Practical Guide to Functional Measurement, New Jersey: Prentice Hall.

Han, C. and Shin, K.G. (1995) Real-Time Communication in Fieldbus Multiaccess Networks, Proceedings of the Real-Time Technology and Applications Symposium, Chicago, Illinois, USA, May 15-17, 1995, IEEE Computer Society Washington, DC, 86-95.

Harrison, P.G. and Patel, N.M. (1993) Performance Modelling of Communication Networks and Computer Architecture, Wokingham: Addison-Wesley Publishing Company.

BIBLIOGRAPHY

HW Server s.r.o.(2005) LIN - Local Interconnect Network [online], available: <http://hw-server.com/docs/lin.html> [accessed 16 June 2008].

Jackman, B. (2008) Class Notes on OSEX/VDX, Waterford Institute of Technology.

Karami, J. and Salahoor, K. (2006) Design and Implementation of an Instructional Foundation Fieldbus-Based Pilot Plant, Proceedings of the International Multi-Conference on Computing in the Global Information Technology, Bucharest, Romania, August 1-3, 2006, IEEE Computer Society Washington, DC, 32-36.

Konak, A., Smith, A. and Kulturel-Konak, S. (2004) New Event-Driven Sampling Techniques for Network Reliability Estimation, Ingalls, R.G., Rossetti, M.D., Smith, J.S. and Peters, B.A., eds., Proceedings of the 2004 Winter Simulation Conference, Washington D.C., USA, December 5-8 2004, , IEEE Computer Society Washington, DC, USA, 224-231.

Kopetz, H. (1998) The Time-Triggered Architecture, The First IEEE International Symposium on Object-Oriented Real-Time Distributed Computing, Kyoto, Japan, April 20 – 22, IEEE Computer Society Washington, DC, USA, 22-29.

Kopetz, H. (1998) The Time-Triggered Model of Computation, 19th IEEE Real-Time Systems Symposium, Madrid, Spain, December 2-4, IEEE Computer Society Washington, DC, USA, 168-177.

Lee, D. and Shi, J.D. (2004) Statistical Analysis for Simulating Schedule Networks, Ingalls, R.G., Rossetti, M.D., Smith, J.S. and Peters, B.A., eds., Proceedings of the 2004 Winter Simulation Conference, Washington D.C., USA, December 5-8 2004, IEEE Computer Society Washington, DC, USA, 1283 – 1289.

Lee, H.J., Kim, S.H., Park, H.S. and Park, B.S. (2008) Discrete Event System Simulation Approach For A Nuclear Facility Operational Analysis, Proceedings of Innovative Production Machines and Systems Conference, July1-14, Whittles Publishing, Dunbeath, Caithness Scotland.

BIBLIOGRAPHY

- Leen, G. and Heffernan, D. (2002) Expanding Automotive Electronic Systems, *Computer*, 35(1), 88-93.
- Leen, G. and Heffernan, D. (2006) Modeling and Verification of a Time-Triggered Networking Protocol, *Proceedings of the International Conference on Networking, International Conference on Systems and International Conference on Mobile Communications and Learning Technologies*, April 23-29, Mauritius, IEEE Computer Society Washington, DC, USA, 178-188.
- Leppla G. (2008) Mapping Requirements to AUTOSAR Software Components, unpublished thesis (M.Sc.), Waterford Institute of Technology.
- Levi, S. and Agrawala, A.K. (1990) *Real Time System Design*, Singapore: McGraw-Hill.
- Lexico Publishing Group, LLC (2008) Dictionary.com [online], available: <http://dictionary.reference.com/> [accessed 4 June 2008].
- Lv, J., Li, T. and Li, X. (2007) Network Traffic Prediction Algorithm and its Practical Application in Real Network, 2007 IFIP Conference on Network and Parallel Computing – Workshops, Dalian, China, September 18-21 2007, IEEE Computer Society Washington, DC, USA, 512-517.
- Lv, J., Li, X., Anh, T.Q. and Li, T. (2006) A New Algorithm for Network Traffic Prediction, *Proceedings of the 11th IEEE Symposium on Computers and Communications*, Pula-Cagliari, Sardinia, Italy, June 26-29 2006, IEEE Computer Society Washington, DC, USA, 1006 - 1012.
- Maier, R., Bauer, G., Stöger, G. and Poledna, S. (2002) Time-Triggered Architecture: A Consistent Computing Platform, *IEEE Micro*, 22(4), 36-45.
- Massey, R. (2001) Introduction to Interrupts [online], available: <http://www.embedded.com/story/OEG20010518S0075> [accessed 29 July 2008].

BIBLIOGRAPHY

Mathworks Automotive Advisory Board (MAAB) (2007), Control Algorithm Modeling Guidelines Using MATLAB, Simulink, and Stateflow Version 2.0.

MB91460 Series Start 91460.ASM Application Note, Langen, Germany.

Mueller, C. (2004) Addressing: the Root of All programming Evils, Proceedings of the 28th Annual International Computer Software and Applications Conference volume 2, Hong Kong, China, September 28-30 2004 , IEEE Computer Society Washington, DC, USA 14-15.

Murphy R. (2009) A Migration Framework from CAN to FlexRay, unpublished thesis (M.Sc.), Waterford Institute of Technology.

Nadlerlinger, A., Pletzer, J., Pree, W. and Templ, J. (2007) Model Driven Development of FlexRay-Baed Systems with the Timing Definition Language (TDL), Proceedings of the Fourth International Workshop on Software Engineering for Automotive Systems, Minneapolis, Minnesota, USA, May 20-26 2007, IEEE Computer Society Washington, DC, USA.

Nossal, R. and Lang, R. (2002) Model-based System Development- an Approach to building X-by-Wire Applications, IEEE Micro, 22(4), IEEE Computer Society Washington, DC, USA, 56-63.

Okabe, N. (2005) Issues of Control Networks when Introducing IP, Proceedings of the 2005 Symposium on Applications and the Internet Workshop Trento, Italy, January 31 – February 4 2005, IEEE Computer Society Washington, DC, USA, 80-83.

Oldfield, J.V. and Dorf, R.C. (1995) Field Programmable Gate Arrays, Canada, John Wiley & Sons, Inc.

Pree, W. and Templ, J.(2007) Forget about FlexRay, FlexRay Product Day, November 29, Fellbach Germany, Carl Hanser Verlag, Münch.

BIBLIOGRAPHY

Peterson, J.L. (1991) Petri Net Theory and the Modelling of Systems, New Jersey: Prentice-Hall.

Phillips, C.L. and Nagle, H.T. (1995) Digital Control Systems Analysis and Design, third edition, New Jersey: Prentice Hall.

Pont, M.J. (2001) Patterns for Time-Triggered Embedded Systems – Building Reliable Applications with the 8051 Family of Microcontrollers, Harlow: Addison-Wesley.

PROFIBUS (2007) PROFIBUS PA Technology and Application, System Description, Karlsruhe: PROFIBUS Nutzerorganisation e.V.

QNX Software Systems Ltd (2004) Media Orientated Systems Transport (MOST), Canada: QNX Software Systems Ltd.

Ray, J. and Koopman, P. (2006) Efficient High hamming Distance CRCs for Embedded Networks, Proceedings of the 2006 International Conference on Dependable Systems and Networks, Philadelphia, Pennsylvania , USA, June 25-28, IEEE Computer Society Washington, DC, USA, 3-12.

Real-Time Systems Group University of Technology Vienna (2002) Specification of the TTP/A-Protocol V2.00, Vienna, Austria: Real-Time Systems Group.

Ripoll, I., Pisa, P., Abeni, L., Gai, P., Lanusse, A., Saez, S. and Privat, B. (2002) WP1 - RTOS State of the Art Analysis [online], available: http://www.mnis.fr/ocera_support/rtos/book1.html [accessed 11 June 2008].

Sangiovanni-Vincentelli, A. and Di Natale, M. (2007) Embedded Systems Design for Automotive Applications, 40(10), 43-51.

Saski, H., Iwasaki, H. and Suda, T. (2004) Simulation of Information Propagation for Vehicles in Physical Communication Network Models, Proceedings of the 2004

BIBLIOGRAPHY

International Symposium on Applications and the Internet Workshops, Tokyo, Japan, January 26-30 2004, IEEE Computer Society Washington, DC, USA, 408-416.

Schedl, A. (2004) OSEKtime – Zeitgesteuerte Erweiterung des OSEK Standards, Time-Triggered Real-Time Operating Systems and Fault Tolerant Communication Layer for Drive-by-Wire Applications.

Schober, C. (2004) Distributed Real-Time Applications, unpublished thesis - diploma thesis, Salzburg University of Applied Sciences and Technologies.

Scott, A.V. and Buchanan, W.J. (2000) Truly Distributed Control Systems using Fieldbus Technology, Proceedings of the Seventh IEEE International Conference on Engineering of Computer Based Systems, Edinburgh, Scotland, UK, April 3-7 2000, IEEE Computer Society Washington, DC, 165 – 173.

Seo, S.H., Park, J.H., Hwang, S.H. and Jeon, J.W. (2006) 3-D Car Simulator for testing ECU Embedded Systems, Proceedings of SICE-ICASE International Joint Conference, Bexco, Busan, Korea, October 18-21 2006, Computer Society Washington, DC, USA, 550-554.

Sharma, A.K. (1998) Programmable Logic Handbook, USA, McGraw-Hill.

Sparachmann, M. (2001) Automatic Generation of Parallel CRC Circuits, IEEE Design & Test of Computers, 18(3), 108-114.

Tang, K.H., Sambamoorthy, M., Lin, P.K. and Kandiah, K. (1996) Chiller plant Control and Efficiency Optimisation, CAN Newsletter, September 2006, 8-16.

Techmer, A. and Leiteinturier (2006) Implementing FlexRay on Silicon, Proceedings of the International Conference on Networking, International Conference on Systems and International Conference on Mobile Communications and Learning Technologies, Piscataway, New Jersey, April 23-29 2006, IEEE Computer Society Washington, DC, USA, 34-39.

BIBLIOGRAPHY

telos EDV Systementwicklung GmbH (2008) MOST Network [online], available: <http://www.telos.info/MOST-R-Network.31.0.html> [accessed 16 June 2008].

The MathWorks, Inc. (2000) Creating Graphical User Interfaces version 1, Massachusetts: The MathWorks, Inc.

The MathWorks, Inc. (2008) SimEvents® Release Notes, Massachusetts: The MathWorks, Inc.

Tindell, K. and Clark, J. (1994) Holistic Schedulability for Distributed Hard Real-Time Systems, Microprocessing and Microprogramming - Euromicro Journal (Special Issue on Parallel Embedded Real-Time Systems) 40, 117-134.

Tracey, N. (2001) Comparing OSEK and OSEK Time.

Valdimarsson, E. (1995) Queueing Analysis for Shared Buffer Switching Networks for Non-Uniform Traffic, Proceedings of the Fourteenth Annual Joint Conference of the IEEE Computer and Communication Society, Boston, Massachusetts, USA, April 2-6, 1995, IEEE Computer Society Washington, DC, USA, 8-15.

Vincent, J-M. (2005) Perfect Simulation of Queueing Networks with Blocking and Rejection, Proceedings of the 2005 Symposium on Applications and the internet Workshops, Trento, Italy, January 31 – February 4 2005, IEEE Computer Society Washington, DC, USA, 268-271.

Walrand, J. (1998) Communication Networks – A First Course, Second Edition, America: WCB/ MacGraw-Hill.

Walsh, J. (2008) Construction Of Vehicle Deterioration Models Based On Driving Style Analysis, unpublished thesis (M.Sc.), Waterford Institute of Technology.

Yin, Q., Jiang, Y., Jiang, S. and Yong Kong, P. (2002) Analysis on General Stochastically Bounded Bursty Traffic for Communication Networks, Proceedings of

BIBLIOGRAPHY

the 27th Annual IEEE Conference on Local Computer Networks, Tampa, FL, USA, November 6-8 2002, IEEE Computer Society Washington, DC, USA, 141-149.

Zanoni, E. and Pavan, P. (1993) Improving the Reliability and Safety of Automotive Electronics, IEEE Micro, 13(1), 30-48.

Zhao, W. and Xia, F. (2006) Design and Simulation of Function Block Based Networked Control Systems, Proceedings of the First International Conference on Innovative Computing, Information and Control, Beijing, China, August 30 - September 1 2006, IEEE Computer Society Washington, DC, USA, 287 – 291.






















Zheng, W., Chong, C., Pinello, C., Kanajan, S. and Sangiovanni-Vincentelli (2005) Extensible and Scalable Time Trigered Scheduling, in Proceedings of the Fifth International Conference on Application of Concurrency to System Design, St Malo, France, June 7-9, 2005, IEEE Computer Society Washington, DC, USA, 132-141.

Section VII:
Appendices

Appendix A:
Model Colour Coding

APPENDIX A

The table below, Table A.1, shows the colour coding values for each element of the model. This was used to distinguish signals from different parts of the model. The table gives the RGB (Red, Green and Blue) values and the HSL (Hue, Saturation and Luminescence) values as well as a sample. The RGB and HSL values are simple do different ways of representing colour in the computers colour space. Both were given only as reference. It is not necessary to have all this information to reproduce these colours.

Layer	Red Value	Green Value	Blue Value	Hue	Saturation	Luminescence	Sample
Top Layer							
Physical Bus	251	253	181	41	227	204	
Host	204	197	245	166	169	208	
Physical Bus							
Data In	182	126	90	16	93	128	
Data Out	254	189	101	23	237	167	
Host							
Driver	126	157	237	149	181	171	
Application	255	209	125	26	240	179	
Communications Controller	163	241	174	86	177	190	
Driver							
To Application	255	209	125	26	240	179	
To Communications Controller	163	241	174	86	177	190	
Application							
To Driver	126	157	237	149	181	171	
Communications Controller							
Data To/From Driver	126	157	237	149	181	171	
Transmit Data	254	189	101	23	237	167	
Receive Data	182	126	90	16	93	128	
Synchronisation	252	139	142	239	228	184	
Media Access Control	104	174	102	79	74	130	
Frame and Symbol Processing	255	255	128	40	240	180	
Protocol Operations Control	187	136	249	178	217	181	
Controller Host Interface	149	191	173	103	59	160	
Synchronisation							
Static Segment	255	128	0	20	240	120	
Dynamic Segment	0	209	209	120	240	98	
Symbol & Network Idle Time	128	128	128	160	0	120	

APPENDIX A

Initialise	107	150	61	59	101	99	
Cycle Count	97	189	252	136	231	164	
Media Access Control							
Data For Transmission	254	189	101	23	237	167	
Frames In	187	136	249	178	217	181	
Slots In	159	150	117	31	43	130	
Frame and Symbol Processing							
Received Data In	182	126	90	16	93	128	
Received Data out	187	136	249	178	217	181	
Cycle Count	97	189	252	136	231	164	
Protocol Operations Control							
Static Slots In	255	128	0	20	240	120	
Mini Slots In	0	209	209	120	240	98	
Cycle Entity	97	189	252	136	231	164	
Data To/From Driver	126	157	237	149	181	171	
Received Data	182	126	90	16	93	128	
Transmit Data	254	189	101	23	237	167	
Slots To Media Access Control	104	174	102	79	74	130	
Global Time Unit	159	150	117	31	43	130	
Message Handler	254	177	214	221	234	203	
Received Message Routing	182	126	90	16	93	128	
Controller Host Interface							
Data To Driver	126	157	237	149	181	171	
Data To Protocol Operations Control	187	136	249	178	217	181	
Message Handler							
Message RAM Initialisation	107	150	61	59	101	99	
Message RAM	192	192	192	160	0	181	
Requests In/Out	126	157	237	149	181	171	
Receive Data	182	126	90	16	93	128	
Transmit Data	254	189	101	23	237	167	
Slots In	159	150	117	31	43	130	
Global Time Unit							
Slots To MAC	104	174	102	79	74	130	
Slots To Message Handler	254	177	214	221	234	203	
Static Slots	255	128	0	20	240	120	
Mini Slots	0	209	209	120	240	98	
Received Data	182	126	90	16	93	128	
Overall Model							
Record Data	165	209	254	140	235	197	

Table A.1: Model colour coding

Appendix B:
Model Variables
& Attributes

APPENDIX B

The tables below, Tables B.1-B.8, marked as input variables indicate values that must be supplied. Tables marked as output are variables that are returned by the model for analytical purposes.

Physical Bus		
	Channle A	Channel_A_Record
	Channle A	Channel_A_Enable
	Channle A	Byte_Prop_Delay
	Channle A	Mini_Time
	Channle B	Channel_B_Record
	Channle B	Channel_B_Enable
	Channle B	Byte_Prop_Delay
	Channle B	Mini_Time
	Additional_Frames	First_Dynamic
	Additional_Frames	Static_Length
	Additional_Frames	Channel_Generated_Frame_A
	Additional_Frames	Latest_TX
	Additional_Frames	Generate_Frame_A_Time
	Additional_Frames	Generate_Frame_B_Time
	Additional_Frames	Channel_Generated_Frame_B
	Additional_Frames	Channel_A_Enable
	Additional_Frames	Channel_B_Enable
	Additional_Frames	CHA_Prop_Delay
	Additional_Frames	Mini_Time
	Additional_Frames	Num_Mini
	Additional_Frames	Average_Dynamic
	Additional_Frames	Variance
	Additional_Frames	CHB_Prop_Delay
	Additional_Frames	Frame_IDs

Table B.1: Physical Bus input workspace variables

FlexRay Node		
	Application	Application_Record
	Application	Generate_Request_Time
	Application	Num_Requests_Generated
	Application	Average_Dynamic
	Application	Channel_Response
	Application	Variance
	Application	Request_Generation
	Application	First_Dynamic
	Application	Channel_Generated_Frame
	Application	Read_Length
	Application	Request_Types

APPENDIX B

	Application	Frame_IDs
	Application	Request_IDs
	Application	Step_Time
	Application	Byte_Processing_Time
	Application	Request_Length
	Application	Frame_ID_Response
	Application	Last_Request
	Application	Frame_Response
	Application	Channel_Response
	Application	Static_Length
	Driver	Driver_Record
	Driver	Driver_Delay

Table B.2: Node input workspace variables

Communications Controller		
	Frame_And_Symbol_Processing	FSP_Record
	Frame_And_Symbol_Processing	Cycle_Filtering_Indicator
	Frame_And_Symbol_Processing	Channel_A_Slot_Filtering
	Frame_And_Symbol_Processing	Channel_B_Slot_Filtering
	Frame_And_Symbol_Processing	FSP_Delay
	Frame_And_Symbol_Processing	Channel_A_FIFO_Slot_Filtering
	Frame_And_Symbol_Processing	Channel_B_FIFO_Slot_Filtering
	Frame_And_Symbol_Processing	Frame_IDs
	Synchronisation	Static_Initialise_Complete
	Synchronisation	Mini_Initialise_Complete
	Synchronisation	Sync_Record
	Synchronisation	Num_Static
	Synchronisation	Static_Time
	Synchronisation	Static_Segment_Time
	Synchronisation	Dynamic_Segment_Time
	Synchronisation	Num_Mini
	Synchronisation	Mini_Time
	Synchronisation	Symbol_NIT_Time
	Controller_Host_Interface	CHI_Record
	Controller_Host_Interface	IBF_Delay
	Controller_Host_Interface	OBF_Delay
	Media_Access_Control	MAC_Record
	Media_Access_Control	MAC_Delay

Table B.3: Communications controller input workspace variables

APPENDIX B

Protocol Operations Control		
	Protocol_Operations_Control	POC_Record
	Global_Time_Unit	Last_Dynamic_Slot
	Global_Time_Unit	Last_Static_Slot
	Global_Time_Unit	Num_Mini
	Global_Time_Unit	Symbol_NIT_Time
	Global_Time_Unit	Mini_Time
	Message_Handler	Message_Init
	Message_Handler	Finished_RAM_Init
	Message_Handler	MSG_Handler_Record
	Message_Handler	Message_RAM_Delay
	Message_RAM	Num_RAM_Locations
	Message_RAM	RAM_Update_Time
	Message_RAM	Single_Shot_Indicator
	FIFO	Num_FIFO_Locations
	FIFO	FIFO_Update_Delay
	FIFO	FIFO_Read_Delay

Table B.4: Protocol operations control input workspace variables

Model Section	Model Block	Output Variables
Physical Bus		
	Additional_Frames	Slot_Num_Generated_Frames_A
	Additional_Frames	Remaining_Generated_Frames_Mini_A
	Additional_Frames	Cycle_Count_Generated_Frames_A
	Additional_Frames	Slot_Num_Generated_Frames_B
	Additional_Frames	Remaining_Generated_Frames_Mini_B
	Additional_Frames	Cycle_Count_Generated_Frames_B
	Additional_Frames	Frame_ID_Generated_Frames_A
	Additional_Frames	Cycle_Code_Generated_Frames_A
	Additional_Frames	Channel_Config_Generated_Frames_A
	Additional_Frames	Data_Generated_Frames_A
	Additional_Frames	Frame_ID_Generated_Frames_B
	Additional_Frames	Cycle_Code_Generated_Frames_B
	Additional_Frames	Channel_Config_Generated_Frames_B
	Additional_Frames	Data_Generated_Frames_B
	Channel_A	Frame_ID_Channel_A_In
	Channel_A	Cycle_Code_A_In
	Channel_A	Channel_Config_A_In
	Channel_A	Data_A_In
	Channel_B	Frame_ID_Channel_B_In
	Channel_B	Cycle_Code_B_In

APPENDIX B

	Channel_B	Channel_Config_B_In
	Channel_B	Data_B_In
	Channel_A	Frame_ID_Channel_A_Out
	Channel_A	Cycle_Code_A_Out
	Channel_A	Channel_Config_A_Out
	Channel_A	Data_A_Out
	Channel_B	Frame_ID_Channel_B_Out
	Channel_B	Cycle_Code_B_Out
	Channel_B	Channel_Config_B_Out
	Channel_B	Data_B_Out
	Channel_A	Slot_Num_A
	Channel_A	Remaining_Mini_A
	Channel_A	Cycle_Count_A
	Channel_B	Slot_Num_B
	Channel_B	Remaining_Mini_B
	Channel_B	Cycle_Count_B

Table B.5: Physical Bus output workspace variables

Model Section	Model Block	Variable
FlexRay Node		
	Application	Cycle_Num_Application_In
	Application	Request_Type_Application_In
	Application	Frame_ID_Application_In
	Application	Data_Driver_Application_In
	Application	Channel_Config_Application_In
	Application	Request_Type_Application_Out
	Application	Frame_ID_Application_Out
	Application	Data_Driver_Application_Out
	Application	Channel_Config_Application_Out
	Driver	Request_Type_Driver_To_ERay_In
	Driver	Frame_ID_Driver_To_ERay_In
	Driver	Data_Driver_To_ERay_In
	Driver	Channel_Config_Driver_To_ERay_In
	Driver	Request_Type_Driver_To_ERay_Out
	Driver	Frame_ID_Driver_To_ERay_Out
	Driver	Data_Driver_To_ERay_Out
	Driver	Channel_Config_Driver_To_ERay_Out
	Driver	Request_Type_ERay_To_Driver_In
	Driver	Frame_ID_ERay_To_Driver_In
	Driver	Data_ERay_To_Driver_In
	Driver	Channel_ERay_To_Driver_In
	Driver	Request_Type_ERay_To_Driver_Out
	Driver	Frame_ID_ERay_To_Driver_Out
	Driver	Data_ERay_To_Driver_Out
	Driver	Channel_ERay_To_Driver_Out
	Driver	Num_CC_Driver

APPENDIX B

	Driver	Average_Wait_CC_Driver
	Driver	Average_CC_Driver_Length
	Driver	Num_Driver_CC
	Driver	Average_Wait_Driver_CC
	Driver	Average_Driver_CC_Length

Table B.6: Node output workspace variables

Model Section	Model Block	Variable
Communications Controller		
	Frame_And_Symbol_Processing	Frame_ID_FSP_A_In
	Frame_And_Symbol_Processing	Cycle_Code_FSP_A_In
	Frame_And_Symbol_Processing	Channel_Config_FSP_A_In
	Frame_And_Symbol_Processing	Data_FSP_A_In
	Frame_And_Symbol_Processing	Frame_ID_FSP_B_In
	Frame_And_Symbol_Processing	Cycle_Code_FSP_B_In
	Frame_And_Symbol_Processing	Channel_Config_FSP_B_In
	Frame_And_Symbol_Processing	Data_FSP_B_In
	Frame_And_Symbol_Processing	Slot_Num_FSP_A_In
	Frame_And_Symbol_Processing	Remaining_Mini_FSP_A_In
	Frame_And_Symbol_Processing	Cycle_Count_FSP_A_In
	Frame_And_Symbol_Processing	Slot_Num_FSP_B_In
	Frame_And_Symbol_Processing	Remaining_Mini_FSP_B_In
	Frame_And_Symbol_Processing	Cycle_Count_FSP_B_In
	Frame_And_Symbol_Processing	Slot_Num_FSP_A_Out
	Frame_And_Symbol_Processing	Remaining_Mini_FSP_A_Out
	Frame_And_Symbol_Processing	Cycle_Count_FSP_A_Out
	Frame_And_Symbol_Processing	Slot_Num_FSP_B_Out
	Frame_And_Symbol_Processing	Remaining_Mini_FSP_B_Out
	Frame_And_Symbol_Processing	Cycle_Count_FSP_B_Out
	Frame_And_Symbol_Processing	Frame_ID_Filter_Frames_A_In
	Frame_And_Symbol_Processing	Cycle_Code_Filter_Frames_A_In
	Frame_And_Symbol_Processing	Channel_Config_Filter_Frames_A_In
	Frame_And_Symbol_Processing	Data_Filter_Frames_A_In
	Frame_And_Symbol_Processing	Frame_ID_Filter_Frames_B_In
	Frame_And_Symbol_Processing	Cycle_Code_Filter_Frames_B_In
	Frame_And_Symbol_Processing	Channel_Config_Filter_Frames_B_In
	Frame_And_Symbol_Processing	Data_Filter_Frames_B_In
	Frame_And_Symbol_Processing	Frame_ID_Filtered_Frames_A_Out
	Frame_And_Symbol_Processing	Cycle_Code_Filtered_Frames_A_Out
	Frame_And_Symbol_Processing	Channel_Config_Filtered_Frames_A_Out
	Frame_And_Symbol_Processing	Data_Filtered_Frames_A_Out
	Frame_And_Symbol_Processing	Frame_ID_Filtered_Frames_B_Out
	Frame_And_Symbol_Processing	Cycle_Code_Filtered_Frames_B_Out
	Frame_And_Symbol_Processing	Channel_Config_Filtered_Frames_B_Out
	Frame_And_Symbol_Processing	Data_Filtered_Frames_B_Out
	Frame_And_Symbol_Processing	Frame_ID_FIFO_Filter_Frames_A_In
	Frame_And_Symbol_Processing	Cycle_Code_FIFO_Filter_Frames_A_In

APPENDIX B

	Frame_And_Symbol_Processing	Channel_Config_FIFO_Filter_Frames_A_In
	Frame_And_Symbol_Processing	Data_FIFO_Filter_Frames_A_In
	Frame_And_Symbol_Processing	Frame_ID_FIFO_Filter_Frames_B_In
	Frame_And_Symbol_Processing	Cycle_Code_FIFO_Filter_Frames_B_In
	Frame_And_Symbol_Processing	Channel_Config_FIFO_Filter_Frames_B_In
	Frame_And_Symbol_Processing	Data_FIFO_Filter_Frames_B_In
	Frame_And_Symbol_Processing	Frame_ID_FIFO_Filter_Frames_A_Out
	Frame_And_Symbol_Processing	Cycle_Code_FIFO_Filter_Frames_A_Out
	Frame_And_Symbol_Processing	Channel_Config_FIFO_Filter_Frames_A_Out
	Frame_And_Symbol_Processing	Data_FIFO_Filter_Frames_A_Out
	Frame_And_Symbol_Processing	Frame_ID_FIFO_Filter_Frames_B_Out
	Frame_And_Symbol_Processing	Cycle_Code_FIFO_Filter_Frames_B_Out
	Frame_And_Symbol_Processing	Channel_Config_FIFO_Filter_Frames_B_Out
	Frame_And_Symbol_Processing	Data_FIFO_Filter_Frames_B_Out
	Frame_And_Symbol_Processing	Frame_ID_Rejected_Frames_A
	Frame_And_Symbol_Processing	Cycle_Code_Rejected_Frames_A
	Frame_And_Symbol_Processing	Channel_Config_Rejected_Frames_A
	Frame_And_Symbol_Processing	Data_Rejected_Frames_A
	Frame_And_Symbol_Processing	Frame_ID_Rejected_Frames_B
	Frame_And_Symbol_Processing	Cycle_Code_Rejected_Frames_B
	Frame_And_Symbol_Processing	Channel_Config_Rejected_Frames_B
	Frame_And_Symbol_Processing	Data_Rejected_Frames_B
	Synchronisation	Slot_Num_Generated
	Synchronisation	Cycle_Num_Generated
	Synchronisation	Mini_Num_Generated_A
	Synchronisation	Mini_Num_Generated_B
	Controller_Host_Interface	Request_Type_Driver_CHI_In
	Controller_Host_Interface	Frame_ID_Driver_CHI_In
	Controller_Host_Interface	Data_Driver_Driver_CHI_In
	Controller_Host_Interface	Channel_Config_Driver_CHI_In
	Controller_Host_Interface	Request_Type_Driver_CHI_Out
	Controller_Host_Interface	Frame_ID_Driver_CHI_Out
	Controller_Host_Interface	Data_Driver_Driver_CHI_Out
	Controller_Host_Interface	Channel_Config_Driver_CHI_Out
	Controller_Host_Interface	Request_Type_CHI_Driver_In
	Controller_Host_Interface	Frame_ID_CHI_Driver_In
	Controller_Host_Interface	Data_Driver_Driver_CHI_In
	Controller_Host_Interface	Channel_Config_CHI_Driver_In
	Controller_Host_Interface	Request_Type_CHI_Driver_Out
	Controller_Host_Interface	Frame_ID_CHI_Driver_Out
	Controller_Host_Interface	Data_Driver_CHI_Driver_Out
	Controller_Host_Interface	Channel_Config_CHI_Driver_Out
	Media_Access_Control	Frame_ID_MAC_A_In
	Media_Access_Control	Cycle_Code_MAC_A_In

APPENDIX B

	Media_Access_Control	Channel_Config_MAC_A_In
	Media_Access_Control	Data_MAC_A_In
	Media_Access_Control	Frame_ID_MAC_B_In
	Media_Access_Control	Cycle_Code_MAC_B_In
	Media_Access_Control	Channel_Config_MAC_B_In
	Media_Access_Control	Data_MAC_B_In
	Media_Access_Control	Frame_ID_MAC_A_Out
	Media_Access_Control	Cycle_Code_MAC_A_Out
	Media_Access_Control	Channel_Config_MAC_A_Out
	Media_Access_Control	Data_MAC_A_Out
	Media_Access_Control	Frame_ID_MAC_B_Out
	Media_Access_Control	Cycle_Code_MAC_B_Out
	Media_Access_Control	Channel_Config_MAC_B_Out
	Media_Access_Control	Data_MAC_B_Out
	Media_Access_Control	Slot_Num_MAC_A
	Media_Access_Control	Remaining_Mini_MAC_A
	Media_Access_Control	Cycle_Count_MAC_A
	Media_Access_Control	Slot_Num_MAC_B
	Media_Access_Control	Remaining_Mini_MAC_B
	Media_Access_Control	Cycle_Count_MAC_B

Table B.7: Communications controller output workspace variables

Model Section	Model Block	Variable
Protocol Operations Control		
	Protocol_Operations_Control	Cycle_Num_POC
	Protocol_Operations_Control	Slot_Num_POC
	Protocol_Operations_Control	Mini_Num_POC_A_In
	Protocol_Operations_Control	Mini_Num_POC_B_In
	Protocol_Operations_Control	Frame_ID_Frame_POC_A_In
	Protocol_Operations_Control	Cycle_Code_Frame_POC_A_In
	Protocol_Operations_Control	Channel_Config_Frame_POC_A_In
	Protocol_Operations_Control	Data_Frame_POC_A_In
	Protocol_Operations_Control	Frame_ID_Frame_POC_B_In
	Protocol_Operations_Control	Cycle_Code_Frame_POC_B_In
	Protocol_Operations_Control	Channel_Config_Frame_POC_B_In
	Protocol_Operations_Control	Data_Frame_POC_B_In
	Protocol_Operations_Control	Frame_ID_FIFO_Frame_POC_A_In
	Protocol_Operations_Control	Cycle_Code_FIFO_Frame_POC_A_In
	Protocol_Operations_Control	Channel_Config_FIFO_Frame_POC_A_In
	Protocol_Operations_Control	Data_FIFO_Frame_POC_A_In
	Protocol_Operations_Control	Frame_ID_FIFO_Frame_POC_B_In
	Protocol_Operations_Control	Cycle_Code_FIFO_Frame_POC_B_In
	Protocol_Operations_Control	Channel_Config_FIFO_Frame_POC_B_In
	Protocol_Operations_Control	Data_FIFO_Frame_POC_B_In
	Protocol_Operations_Control	Request_Type_POC_In
	Protocol_Operations_Control	Frame_ID_POC_In

APPENDIX B

	Protocol_Operations_Control	Data_Driver_POC_In
	Protocol_Operations_Control	Channel_Config_POC_In
	Protocol_Operations_Control	Slot_Num_POC_A
	Protocol_Operations_Control	Remaining_Mini_POC_A
	Protocol_Operations_Control	Cycle_Count_POC_A
	Protocol_Operations_Control	Slot_Num_POC_B
	Protocol_Operations_Control	Remaining_Mini_POC_B
	Protocol_Operations_Control	Cycle_Count_POC_B
	Protocol_Operations_Control	Frame_ID_Frame_POC_A_Out
	Protocol_Operations_Control	Cycle_Code_Frame_POC_A_Out
	Protocol_Operations_Control	Channel_Config_Frame_POC_A_Out
	Protocol_Operations_Control	Data_Frame_POC_A_Out
	Protocol_Operations_Control	Frame_ID_Frame_POC_B_Out
	Protocol_Operations_Control	Cycle_Code_Frame_POC_B_Out
	Protocol_Operations_Control	Channel_Config_Frame_POC_B_Out
	Protocol_Operations_Control	Data_Frame_POC_B_Out
	Protocol_Operations_Control	Request_Type_POC_Out
	Protocol_Operations_Control	Frame_ID_POC_Out
	Protocol_Operations_Control	Data_Driver_POC_Out
	Protocol_Operations_Control	Channel_Config_POC_Out
	Global_Time_Unit	Slot_Num_GTU
	Global_Time_Unit	Mini_Num_GTU_A_In
	Global_Time_Unit	Mini_Num_GTU_B_In
	Global_Time_Unit	Cycle_Num_GTU
	Global_Time_Unit	Slot_Num_Bus_Slot_A_GTU_In
	Global_Time_Unit	Slot_Num_Bus_Slot_B_GTU_In
	Global_Time_Unit	Slot_Num_GTU_A
	Global_Time_Unit	Remaining_Mini_GTU_A
	Global_Time_Unit	Cycle_Count_GTU_A
	Global_Time_Unit	Slot_Num_GTU_B
	Global_Time_Unit	Remaining_Mini_GTU_B
	Global_Time_Unit	Cycle_Count_GTU_B
	Message_Handler	Request_Type_MSG_Handler_A_In
	Message_Handler	Frame_ID_MSG_Handler_A_In
	Message_Handler	Data_Driver_MSG_Handler_A_In
	Message_Handler	Channel_Config_MSG_Handler_A_In
	Message_Handler	Request_Type_MSG_Handler_B_In
	Message_Handler	Frame_ID_MSG_Handler_B_In
	Message_Handler	Data_Driver_MSG_Handler_B_In
	Message_Handler	Channel_Config_MSG_Handler_B_In
	Message_Handler	Request_Type_MSG_Handler_In
	Message_Handler	Frame_ID_MSG_Handler_In
	Message_Handler	Data_Driver_MSG_Handler_In
	Message_Handler	Channel_Config_MSG_Handler_In
	Message_Handler	Request_Type_FIFO_Frame_MSG_Handler_A_In

APPENDIX B

	Message_Handler	Frame_ID_FIFO_Frame_MSG_Handler_A_In
	Message_Handler	Data_Driver_FIFO_Frame_MSG_Handler_A_In
	Message_Handler	Channel_Config_FIFO_Frame_MSG_Handler_A_In
	Message_Handler	Request_Type_FIFO_Frame_MSG_Handler_B_In
	Message_Handler	Frame_ID_FIFO_Frame_MSG_Handler_B_In
	Message_Handler	Data_Driver_FIFO_Frame_MSG_Handler_B_In
	Message_Handler	Channel_Config_FIFO_Frame_MSG_Handler_B_In
	Message_Handler	Frame_ID_MSG_Handler_A_Out
	Message_Handler	Cycle_Code_MSG_Handler_A_Out
	Message_Handler	Channel_Config_MSG_Handler_A_Out
	Message_Handler	Data_MSG_Handler_A_Out
	Message_Handler	Frame_ID_MSG_Handler_B_Out
	Message_Handler	Cycle_Code_MSG_Handler_B_Out
	Message_Handler	Channel_Config_MSG_Handler_B_Out
	Message_Handler	Data_MSG_Handler_B_Out
	Message_Handler	Request_Type_MSG_Handler_Out
	Message_Handler	Frame_ID_MSG_Handler_Out
	Message_Handler	Data_Driver_MSG_Handler_Out
	Message_Handler	Channel_Config_MSG_Handler_Out
	Message_RAM	Request_Type_RAM_In
	Message_RAM	Frame_ID_RAM_In
	Message_RAM	Data_Driver_RAM_In
	Message_RAM	Channel_Config_RAM_In
	Message_RAM	Request_Type_RAM_Out
	Message_RAM	Frame_ID_RAM_Out
	Message_RAM	Data_Driver_RAM_Out
	Message_RAM	Channel_Config_RAM_Out
	Message_RAM	Request_Type_FIFO_In
	Message_RAM	Frame_ID_FIFO_In
	Message_RAM	Data_Driver_FIFO_In
	Message_RAM	Channel_Config_FIFO_In
	Message_RAM	Slot_Num_RAM
	Message_RAM	Remaining_Mini_RAM
	Message_RAM	Cycle_Count_RAM
	Message_RAM	Frame_ID_RAM_Out
	Message_RAM	Cycle_Code_RAM_Out
	Message_RAM	Channel_Config_RAM_Out
	Message_RAM	Data_RAM_Out
	Message_RAM	Num_RAM_Messages
	Message_RAM	Average_Wait_Time_RAM
	Message_RAM	Average_RAM_Queue_Length
	FIFO	Num_FIFO_Messages
	FIFO	Average_Wait_Time_FIFO
	FIFO	Average_FIFO_Queue_Length

Table B.8: Protocol operations control output workspace variables

APPENDIX B

The following table, Table B.9, describes the attributes of the various types of entities used within the model.

Type	Attribute	Attribute Number	Notes
Slot			
	Slot_No	1	
	Remaining_Mini	2	Used to calculate the remaining transmission time in the physical bus.
	Cycle_Count	3	
Requests			
	Request_Type	1	1 = get frame for transmission. 2 = get frame for the host. 3 = update frame attributes. 4 = read from the FIFO
	Frame_ID	2	The frame to be updated, transmitted or sent to the host.
	Data	3	The update data if any.
	Channel_Config	4	The channel the message is to be transmitted if necessary.
	TX_Type	1	1 = The frame should be transmitted. This is used when updating a message buffer for transmission. 2 = the frame was received and should not be transmitted.
Frames			
	Frame_ID	1	
	Cycle_Code	2	Used for filtering if desired.
	Channel_Config	3	1 = transmit/receive on A. 2 = transmit/receive on B. 3 = transmit/receive on A & B. This is the same for Request Entities.
	Data	4	
	TX_Type	1	1 = The frame should be transmitted. 2 = the frame was received and should not be transmitted.
Static Slots			
	Slot_Number	1	
	Remaining_Mini	2	
Mini Slots			
	Mini_Slot_Number	1	
Cycle Count			
	Cycle_Count_Number	1	

Table B.9: Entity attributes

Appendix C:
The Model

Source Code

Appendix D:
Technical Papers

**2008 IEEE International Symposium on
Industrial Electronics,
30 June - 2 July 2008,
Cambridge, United Kingdom**

**‘An Introduction to FlexRay as an Industrial Network’
Robert Shaw, Brendan Jackman**

An Introduction to FlexRay as an Industrial Network

Robert Shaw, Brendan Jackman
Automotive Control Group,
Waterford Institute of Technology,
Waterford,
Ireland.

E-mail: rshaw@wit.ie, bjackman@wit.ie
Website: <http://www.wit.ie/automotive>

Abstract

The FlexRay Protocol was developed by the FlexRay consortium which was started when BMW and DaimlerChrysler worked together to create a new network scheme that would suit their current and future needs. They were soon joined by other companies such as Bosch and Phillips [1]. FlexRay is set to become widely used in the automotive industry where it will replace or support existing networking schemes such as CAN. It has already been implemented by BMW in the 2006 X5 [2]. This paper will introduce the key features of the FlexRay protocol and where relevant it will be compared it to CAN and the existing Fieldbus technologies such as PROFIBUS and Foundation Fieldbus.

1. Introduction

The core concept of the FlexRay protocol is a time triggered approach to network communications. This is a different approach to some earlier successful networking schemes. For instance CAN (controller area network) was first developed for use in the automotive industry but was found to be useful in other areas such as industrial control applications [3]. The CAN networking scheme uses a priority driven bus arbitration system. This means that a message with a higher priority message ID will be given access to the network if a lower priority message is also looking for access to the bus. The resulting message transmission delays can lead to problems for safety systems and because of this a TDMA (time division multiple access) method was chosen for the FlexRay protocol [4].

The TDMA scheme used by FlexRay has some similarities to Fieldbus communication systems. With the Foundation Fieldbus a repeating communication cycle determines which node may transmit using a master-slave networking scheme. This ensures that only one node may have access to the physical bus at any one time [5][6][7].

2. FlexRay in Automation

FlexRay is an option for upgrading existing network systems using CAN in the automotive industry as well as other industrial control applications. It could also be used for new applications in industrial automation, where safety and reliability in a work environment is of utmost importance, due to its deterministic approach to communication of the messages. This is helped by the use of a two channel topology where each channel is able to work independently, but the two channels can also be used to communicate the same information and as such has built in redundancy.

The FlexRay protocol has been designed to carry information at a rate of 10Mbits/s over each of its two channels while CAN has a data rate of 1Mbit/s [3][8]. This means that an equivalent data rate of 20Mbits/s can be achieved which is twenty times that of a CAN based system. The high bit rate of FlexRay systems makes it suitable as the basis of a network backbone even where CAN is already in use.

A Fieldbus network has a data rate of around 31.5Kbits/s. When the data rate is compared to that of FlexRay there emerges a view that FlexRay, if used with an existing CAN system, could take on a similar role of the HSE communication role used in Foundation Fieldbus [5][6][7].

3. Network Topology

The FlexRay protocol defines a two channel network, channel A and channel B. A node can be attached to one or both of these channels. If a node is attached to a single channel it does not matter if it is channel A or channel B.

The FlexRay protocol allows for various bus topologies. These can be a point to point connection, passive star, linear passive bus, active star network, cascaded active stars, hybrid

APPENDIX D

topologies and dual channel topologies. The FlexRay protocol will support hybrid topologies as long as the limits of each topology which makes up the hybrid topology (i.e. the star and bus topologies) are not exceeded [8][9]. Fig 1 [8]

shows the possible connections in a dual channel configuration. Other possibilities for the network configuration are to have each channel connected in a different way as shown in Fig 2 [8].

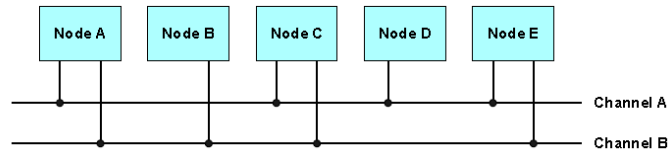


Fig 1. Dual channel configurations

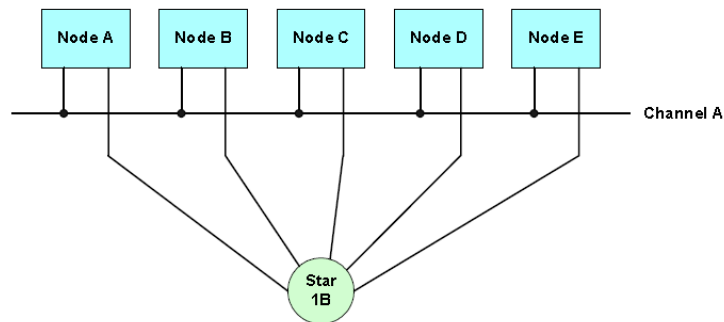


Fig 2. Dual channel implemented as different network types

In Fig 2 channel A is implemented as a bus while channel B is a star topology. The possible combinations of FlexRay topologies make it a very adaptive and flexible system which can be designed to suit various applications.

4. FlexRay Hardware

Each node has a communication controller, a host, a power supply unit and two bus drivers, one for each channel. Fig 3 [8] shows the logical connections of each element.

The host handles the applications of the system while the FlexRay protocol is handled by the communications controller. The bus driver is used to read and write data to the physical medium over which the data is transmitted. In sleep mode it also has the ability to start the wakeup procedure if it detects a wakeup symbol. The communications controller will mainly handle the framing of data and the checking of received data to ensure it was uncorrupted before passing it to the host. The host and communications controller share information such as control information and payload data from the host, while the communication controller relays status information and data received. The host interface to the bus driver allows it to change the operation of the bus driver as well as read status and error flags.

5. Media Access Control

CAN uses a serial bus priority driven networking scheme but allows for a time triggered communications using a higher protocol such as TTCAN [10]. This means that in a basic CAN system if any two nodes wish to transmit data at the same time, the message with the higher priority can transmit while the other message must wait. In contrast the FlexRay protocol uses a TDMA approach and also allows for a node to send frames in a dynamic way. To do this the protocol defines a recurring cycle called the communications cycle. This cycle has the same format and is of the same time length each time it occurs and in the case of FlexRay is divided into four sections: the static segment, dynamic segment, the symbol window and the network idle time. Fig 4 [8] shows a breakdown of the communication into various sections. Each section is then also broken down into its different slots.

The communications protocol of Fieldbus technologies operates in a very similar way to that of FlexRay. However in Fieldbus technologies there is one master node that controls the scheduling of the transmission. In FlexRay each node has its own view of the global time and will only transmit in allocated slots; this means that the network is not dependant on any one node to maintain the communication schedule. The Fieldbus configuration means that the network is liable to fail if the master node fails. In Fieldbus

APPENDIX D

technologies however there is redundancy designed into the devices. This reduces the probability of an entire network failure [5]. In section 6 the method that each FlexRay node uses to determine the current time is outlined.

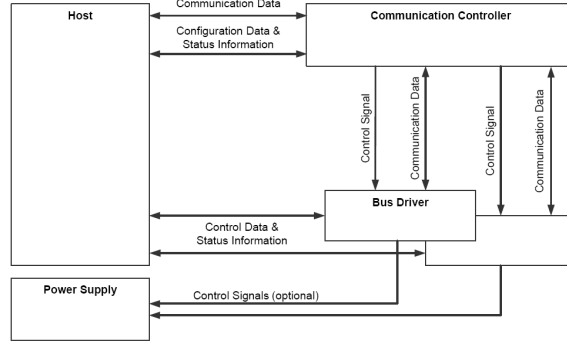


Fig 3. Logical connections

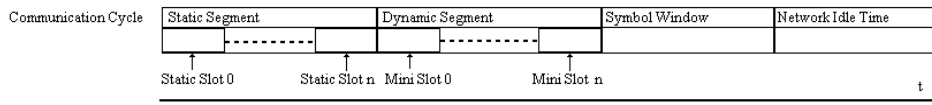


Fig 4. Communication cycle

5.1 TDMA

In a TDMA system the communication cycle is broken down into smaller time segments referred to as slots. The duration of the slots in the static segment are the same. The slots are assigned to a given communication node so that in every communication cycle only that node can transmit at that time. It should be noted that FlexRay does provide a cycle multiplexing system so that information can be sent out every odd cycle for example. This allows another message to be sent in that slot during the even communications cycles and again this message would also be set to that slot in that multiple of the communication cycles. Also a node may get more than one slot per segment depending on the setup of the system and the need to send different messages.

This approach to message arbitration leads to a very deterministic networking scheme making it very suitable for monitoring and safety systems applications.

5.2 Static Segment

The static segment is broken down into smaller sections called static slots. Every static slot is of the same duration. During transmission each slot is assigned to a specific message and only that message can transmit during that slot time.

5.3 Dynamic Segment

The Dynamic segment is an optional section of the communication cycle. It is broken down into smaller sections known as minislots. If a node wishes to communicate it must wait until its minislot comes around. If no transmission occurs after a given period the minislot counter is incremented and the node with the next message/frame id may begin transmission of data. The data to be sent will only be sent if there is enough time left in the dynamic segment. In this way the dynamic segment is priority driven with the message with the lowest ID having the highest priority, just like CAN.

5.4 Symbol Window

A symbol is used to signal a need to wake up a cluster amongst other things. This depends on the symbol sent and the status of the controller at the time. Within the symbol window a single symbol may be sent. If there is more than one symbol to be sent then a higher level protocol must determine which symbol gets priority as the FlexRay protocol provides no arbitration for the symbol window.

5.5 Network Idle Time

The network idle time is used to calculate clock adjustments and correct the node's view of the global time. It also performs communication

APPENDIX D

specific tasks and uses up the remaining time of the communication cycle.

6. Timing

In FlexRay it is important for every node to share the same view of time. This is due to the fact that messages are sent at specific times and so

if there is no global view of time then errors can occur.

To achieve a global view of time each node derives, from the oscillator, a value known as a microtick. This value will vary from node to node. The next level of time is called a

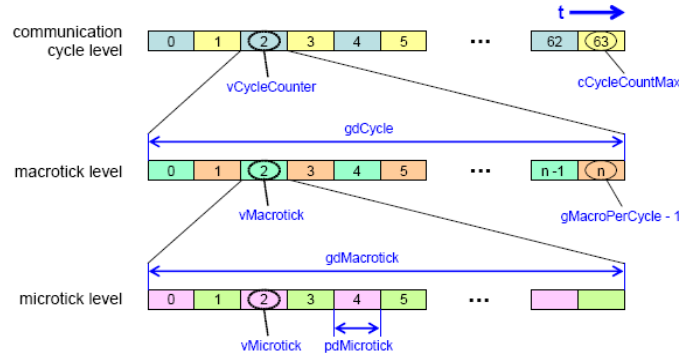


Fig 5. Timing hierarchy

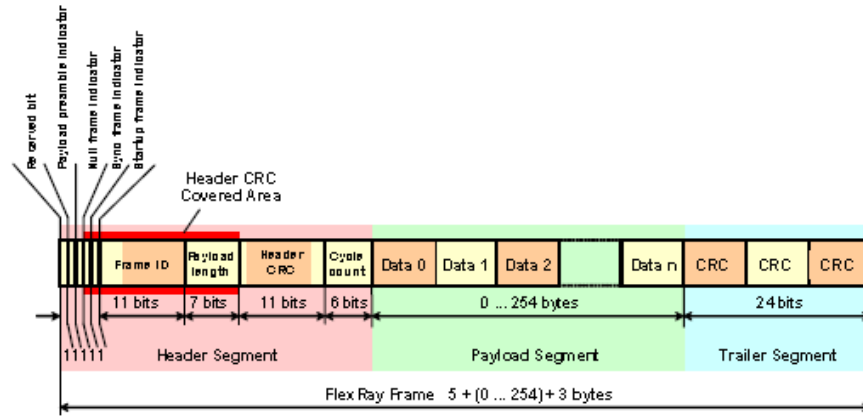


Fig 6. FlexRay frame format

macrotick and is made up of a given number of microticks. This number will vary from node to node so that the duration of a macrotick is the same length throughout the network. Each node will then see the communication cycle in terms of macroticks and this should be the same for all nodes in the network. Fig 5 shows the breakdown of the timing in a FlexRay system.

The nodes in a network will still tend to drift away from one another in terms of their view of the global time. As such the FlexRay protocol defines a clock synchronisation method to keep the networks timing the same within a given tolerance.

7. Frame Format

The frame of a FlexRay message is broken down into 3 sections: the header, payload and trailer section as seen in Fig 6. When compared to the CAN frame format, both standard format and extended format, it can be seen that the

frame format of FlexRay is much larger. This is partly due to the extra error checking.

The header section contains status information such as status bits indicating if the frame is a null frame, i.e. contains no payload data, or if the frame should be used for clock synchronisation. There are also bits to indicate the length of the payload transmitted and cyclic redundancy check (CRC) bits so the receiver can determine if the header was received correctly.

The payload contains the data to be transmitted over the network. The payload may also be used to transmit more frame information as an option and this would be indicated in the header of the frame. The payload length can vary from 0 to 254 bytes. When compared to the 0 to 8 bytes in a CAN frame this is a significant improvement.

The trailer section contains a 24 bit CRC that is calculated over the payload and header sections. This is used by the receiving node again

APPENDIX D

to determine if the frame was received without any errors.

8. FlexRay vs. CAN vs. Fieldbus Summary

The Following table, Table I, compares the feature of FlexRay, CAN and Fieldbus technologies. The use of FlexRay systems in automation could be a useful tool. However when designing a communication system the following trade-offs between FlexRay and other communication protocols should be considered; there would be no point in implementing a FlexRay system if CAN would be sufficient. Likewise if a high data rate is required then FlexRay may be a more suitable scheme to use.

Table I
FlexRay vs. CAN vs. Fieldbus

	FlexRay	CAN	Fieldbus
Communication Method	TDMA but allows for a dynamic communication as standard	Dynamic arbitration on the bus, allowing for a time triggered approach using a higher protocol	Master -Slave configuration where a master determines who can transmit data
Data Rate	10Mbits/s on two channels giving a combined total of 20Mbits/s	1Mbit/s	Values in the kbits/s range between slaves but increasing to up to 100Mbits/s for applications such as the HSE connection of Foundation Fieldbus
Number of transmission bytes	0-254	0-8	1-244 data bytes with between 9 and 11 control bytes for PROFIBUS-PA
Number of communication channels	2 – second channel can be used as a redundant channel	1	1
Error Checking	11 bit header CRC & 24 bit frame CRC	15 bit CRC	1 byte frame check sequence in PROFIBUS-PA
Determinism	TDMA leads to deterministic behaviour	No, unless a higher protocol is used	Master -Slave configuration leads to a predictable pattern
Complexity	High	Low	High
Areas of Use	New protocol means less widely used but high levels of research for future applications	CAN widely used in automation and automotive applications	Widely used for automation purposes

9. Automation Examples

I. Automation Example: Environment Control

An example of where a FlexRay system could be used is in environmental control of buildings. This could have an important role in

manufacturing companies in the future due to the trend of reducing the reliance on fossil fuels to help reduce pollution along with stricter emission laws. This is clear from the Kyoto agreement where countries from around the world are looking to reduce their emissions to 5% above that of their 1990 levels by 2012 [11]. This is compounded by the need of companies to reduce manufacturing costs and with the increasing price of fossil fuels as these also have an effect on their production costs, a more efficient environmental control system could help reduce overall costs. Reference [12] is a CAN based controller designed to attempt to maximise the efficiency of an air conditioning system and thus to minimise the cost of running such a system.

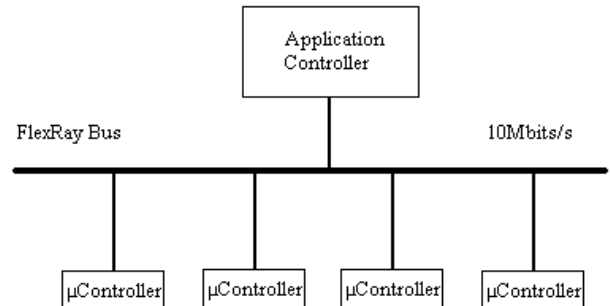


Fig 7. Block diagram for cooler example

This idea can be extended to be used by a FlexRay system. The idea behind this would be a main controller connected to various sensors and actuators over a FlexRay bus. This is illustrated in Figure 7 with the main controller having the ability to receive data from sensors and to send control signals down to the actuators. The diagram illustrates the sensors and actuators as just the microcontroller that would be connected to them.

Due to the deterministic nature of FlexRay this type of system could be set up to monitor or activate the various nodes to increase the efficiency throughout the plant. The sensors placed throughout the plant, both internally and externally, can be used together to determine the best course of action based on internal and external environmental conditions [12]. The high data rate would also mean that more nodes could be serviced in a given time and with bigger possible data payloads information from a given area, such as air temperature and humidity, could be combined into one packet for further efficiency. This could also mean that more sensors could be serviced by a single controller.

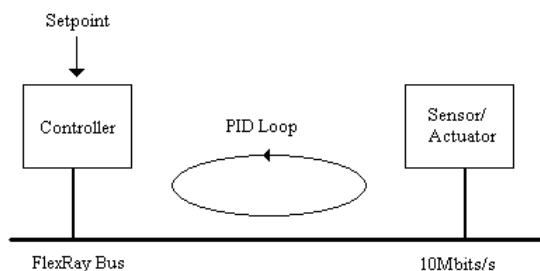


Fig 8. Remote PID loop control

II. Automation Example: Remote PID Control

Example 2 takes the idea of a controller further by using FlexRay to implement a PID controller. Again the high data rates will mean that information will arrive at the controller relatively quickly. The controller will again be able to monitor the incoming data and calculate suitable control signals using a PID algorithm. Again the deterministic nature of FlexRay lends itself to this type of application due to PID needing a regular time base for the calculations and FlexRay only sending data out at predetermined points, e.g. every 3 milliseconds. Figure 8 is a basic block diagram of a FlexRay based PID system. The FlexRay specifications don't specify a particular connection type but usually it is a shielded or unshielded twisted pair with a maximum distance of 24m between nodes [13]. However there is research ongoing into the use of fibre optic cabling. This would make it very suitable in some manufacturing applications where noise is an issue or where the environment may become hazardous in the presence of electricity.

10. Conclusion

The FlexRay protocol developed by the FlexRay consortium has already found applications in the automotive industry and looks set to become the network scheme favoured especially in x-by-wire applications and other safety critical systems. There is on-going research into the migration from CAN based systems to FlexRay based systems and as such the protocol could find itself being used in many areas outside the automotive industry. With its deterministic time-triggered approach and the high data rates achievable it is also suitable for safety and control applications. This paper has briefly introduced the FlexRay protocol. The protocol has further defined areas of the network scheme such as frame and symbol coding/decoding as well as start-up of and integration of a node into a network that are detailed in the FlexRay protocol specification.

Acknowledgements

We would like to thank fellow research group members Gareth Leppla and Richard Murphy for their constructive comments and suggestions while writing this paper.

References

- [1] FlexRay Consortium (2007) about FlexRay [online], available: <http://www.flexray.com/index.php?sid=254188fe2bd59eb7108227f0adea90f5&pid=80&lang=de> [accessed 19 Oct 2007].
- [2] BMW Manufacturing Co. (2006) THE NEW BMW X5 Perfect Blend of Driving Dynamics, Functionality and Exclusivity [press release], 8 August, available: http://www.bmwusfactory.com/media_center/releases/release.asp?intReleaseNum=209&strYear=2006 [accessed 2 October 2007].
- [3] Schofield, M. (2006) Controller Area Network [online], available: <http://www.mjschofield.com/index.htm> [accessed 30 Oct 2007].
- [4] EPN online (2005) Drive-By-Wire: Electronics Vs Mechanical Engineering [online], available: <http://www.epn-online.com/page/16409/drive-by-wire--electronics-vs-mechanical-engineering.html> [accessed 31 Oct 2007].
- [5] PROFIBUS (2007) PROFIBUS PA Technology and Application, System Description, Karlsruhe: PROFIBUS Nutzerorganisation e.V.
- [6] Samson (2000) Foundation Fieldbus Technical Information, Part 4 Communication, Frankfurt: Samson AG.
- [7] Samson (1999) PROFIBUS-PA Technical Information, Part 4 Communication, Frankfurt: Samson AG.
- [8] FlexRay Consortium (2005) FlexRay Communication System Protocol Specification, Version 2.1 Revision A, Stuttgart: FlexRay Consortium GbR.
- [9] FlexRay Consortium (2006) FlexRay Communications System Electrical Physical Layer Specification, Version 2.1 Revision B, Stuttgart: FlexRay Consortium GbR.
- [10] CiA (2007) Controller Area Network (CAN) [online], available: <http://www.can-cia.org/> [accessed 30 Oct 2007].
- [11] Airtricity® Holdings Limited (2007) Kyoto Protocol [online], available: http://www.airtricity.com/ireland/environment/kyoto_protocol/ [accessed 19 November 2007].

APPENDIX D

- [12] Tang, K.H., Sambamoorthy, M., Lin, P.K. and Kandiah, K. (1996) Chiller plant Control and Efficiency Optimisation, CAN Newsletter, September 2006, 8-16.
- [13] FlexRay Consortium (2006) FlexRay Communications System Electrical Physical Layer Application Notes, Version 2.1 Revision B, Stuttgart: FlexRay Consortium GbR.