High-Fidelity Simulation Model of a Dual FIFO CAN Stack

HIGH-FIDELITY SIMULATION MODEL OF A DUAL FIFO CAN STACK

BY

ZHIZHAO QIAN, B.Eng

A THESIS

SUBMITTED TO THE DEPARTMENT OF COMPUTING AND SOFTWARE

AND THE SCHOOL OF GRADUATE STUDIES

OF MCMASTER UNIVERSITY

IN PARTIAL FULFILMENT OF THE REQUIREMENTS

FOR THE DEGREE OF

Master of Applied Science

Master of Applied Science (2018) McMaster University

(Computing and Software) Hamilton, Ontario, Canada

TITLE:	High-Fidelity Simulation Model of a Dual FIFO CAN Stack
AUTHOR:	Zhizhao Qian B.Eng, (Mechatronics Engineering) McMaster University, Hamilton, Ontario, Canada
SUPERVISOR:	Dr. Mark Lawford and Dr. Alan Wassyng
NUMBER OF PAGES:	xvi, 182

Abstract

This thesis presents a simulation model for a Control Area Network (CAN) software stack, the Dual FIFO CAN (DFC) stack, and a method for identifying and incorporating the details of the host environment (hardware setup, operating system, *etc.*) into the implementation of the simulation model in order to achieve a high level of fidelity. The method enable the simulation model to produce more realistic simulation results that are close to real-life experiments of the target system compared to existing commercial and academic simulation tools, which mostly ignore the system details

The simulation model is implemented based on the specification documents of the DFC stack as well as knowledge gained from real-life experiments about the DFC stack and its host environment, a dual-core Electric Control Unit (ECU) hardware test bench that runs a real-time operating system (RTOS). Like the actual DFC stack, the simulation model offers features such as dual non-preemptive FIFO transmit queues and TX buffers, and reserved slots in the queues for higher-priority messages. By using the method introduced in this research, the simulation model also offers options, once enabled and configured with proper parameters, for simulating a host environment that has effects on the behaviors of the modeled CAN stack. And these features are not fully available in existing commercial and academic simulation tools.

The model provides internal calibration values of the DFC stack as configurable parameters to the user, making it easy to customize the simulation. Configurable calibration values includes the total number of slots in the transmit FIFO queues, number of reserved slots in the queues, transmit-rate thresholds that decide to which transmit queue a message is routed and whether a message is eligible to enter the reserved slots of the queues, and together they determine the queuing behaviors of the DFC stack. The options for simulating a host environment (an ECU on a CAN network in a modern vehicle, for instance) is capable of recreating the timing effects (delays, jitters or other effects due to the processing load, physical limitation and internal implementation) of the target host environment on the simulation results. Both deterministic (constant values, *etc.*) and/or statistical (probability distributions, *etc.*) models can be used to configure each single timing effect from the simulated host environment.

The simulation model is also automated to transmit a set of customized transmit message (configurable message ID, DLC, period and internal transmission priority) and process simulation results according to the purpose of the simulation (statistical analysis, plots of data, *etc*). These features make it possible for the simulation model to be used not only to simulate various customized simulation scenarios, but also for different purposes in various stages of the development process, for instance, a pre-experiment simulation run before a test bench experiment to test the correctness of the calibrations and predict the possible outcomes of the experiment, or, simulations for confirmation purposes in order validate the test bench data after the test experiment. The model is compatible with typical modeling, simulation and development environments as it is implemented in MATLAB SimEvents environment, which works

with third-party CAN development tools such as Vector CANoe. It is also designed to work with the high-fidelity model of the Vector CAN protocol stack from Whinton (2016).

Acknowledgments

McMaster University

Mark Lawford

Alan Wassyng

Lucian Patcas

Grant Whinton

Douglas Down

General Motors

Sudhakaran Maydiga

Ramesh S.

Michael A. Turley

Paolo Giusto

Contents

Abstract **iii**

Acknowledgments **vi**

1 Introduction **1**

1.1 The problem 2

1.2 Purpose 3

1.3 Contributions 5

1.4 Organization of the Document 5

2 Preliminaries **6**

2.1 Controller Area Network (CAN) 6

2.1.1 Physical Components 6

2.1.2 Data Format 8

2.1.3 Arbitration 9

2.1.4 Bit Stuffing 11

2.2 Related Work 12

2.3 Existing Tools 13

2.4 MathWorks Software 16

2.4.1 MATLAB 16
2.4.2 Simulink 17
2.4.3 SimEvents 17

3 The Dual FIFO CAN Stack **20**
3.1 Overview of Dual FIFO Queue CAN Stack 20
3.1.1 CAN Handler 22
3.1.2 CAN Device Driver 23
3.2 Message Transmission 24
3.2.1 Message Enqueuing 25
3.2.2 Message Dequeuing 29

4 Hardware Test Bench and Experiment Data **30**
4.1 The Hardware Test Bench 30
4.1.1 The DFC stack and Operating System Scheduling 31
4.1.2 Internal transmit priority of TX Messages 32
4.1.3 Experiment Scenarios and Captured Experiment Data 33
4.2 Observations about Test Bench Behavior and Bench Data 41
4.2.1 Message Enqueue Time 42
4.2.2 Inter-TX-Task Delay 46
4.2.3 Measurement Jitter in Message Bus Times 52

5 The Simulation Model **58**
5.1 The Simulation Script 59
5.1.1 Configurable Input Parameters 59
5.1.2 Simulation Instructions 63

5.1.3 Supporting Scripts 64
5.2 The SimEvents Model 64
5.2.1 Global Data Store Memory Blocks 67
5.2.2 Resource Management Subsystem Blocks 68
5.2.3 The Message Arrivals Subsystem Block 72
5.2.4 The Queuing and Transmission Subsystem Block 77
5.2.5 Return of Successfully Transmitted Messages 95
5.2.6 Simulation Data Logging 97

6 Results and Validation **102**
6.1 Simulations in the Ideal Implementation 106
6.1.1 The 18-75-3 Simulation 109
6.1.2 The 19-50-5 Simulation 117
6.1.3 The 19-50-7 Simulation 125
6.2 Simulations in Practical Environment 134
6.2.1 Statistical Models of the Regular and Irregular Enqueue Times 137
6.2.2 Statistical Model of the Inter-TX-Task Delays 143
6.2.3 Statistical Model of the Measurement Jitter 147
6.2.4 Transmit Rates of Messages 150
6.2.5 Output Message Sequence on the CAN Bus 163
6.3 Summary 174

7 Conclusions **178**
7.1 Future Research 179

List of Figures

2.1 Typical Hardware Components of a CAN Network. 7

2.2 An Example of Arbitration between Three Messages. 10

2.3 An Example of Bit Stuffing on CAN data stream. 11

3.1 Dual FIFO Queue CAN Stack Abstraction Layers 21

3.2 Components in the Layers of Dual FIFO CAN stack. 22

3.3 The Basic Flow of a Transmit Message in the DFC Stack. 25

4.1 The DFC Stack and Dual Core ECU 32

4.2 Calibration Values in the 2 Experiment Configurations of the DFC Stack. 34

4.3 The Message IDs and Nominal Transmit Rates of Transmit Messages in Each Message Set, Listed in the Order of Internal Transmission Priority. 36

4.4 Unique Characteristics of the Transmit Message Sets. 37

4.5 A Section of the Parsed Trace Data of a Test Bench Experiment. . . . 40

4.6 Regular and Irregular Enqueue Times 43

4.7 Impact of Regular Enqueue Times in Message Output Sequence . . . 44

4.8 Impact of Irregular Enqueue Times on Message Output Sequence . . 45

4.9 Time of Inactivity in-between TX Tasks 47

4.10 Profiled $t_{i,1}$ Between 6.25ms TX Task and 12.5ms TX Task 48

4.11 Profiled $t_{i,2}$ Between 12.5ms TX Task and 25ms TX Task 49

4.12 Parameters for Gaussian Distribution Statistical Models of $t_{i,1}$ and $t_{i,2}$. 50
4.13 Effect of Non-zero Inter-TX-Task Delay($t_{i,2}$) on Message Enqueuing . 51
4.14 A Section of the Bus Time Comparison Chart. 54
4.15 The Actual Bus Time and the Measurement Jitter in the Recorded Bus Time of a Message. 55
4.16 The Possible Components of the Measurement Jitter 56
4.17 Measurement Jitter's Impact on Message Throughput 57
5.1 Definition of Message Set 3 in the Simulation Script. 61
5.2 The Root Level Components of the SimEvents Model 65
5.3 General Activity Flow of a Message in the DFC Stack 66
5.4 Configurable Parameters of Resource Acquire Block 70
5.5 The Message Entity Generation Mechanism for the 6.25ms TX Task. 72
5.6 The Structure inside the Message Arrivals Block. 74
5.7 The Mechanism that Updates timer and Decides if a Message is Pending for Transmission. 76
5.8 Mechanism that Divides Transmit-Pending Messages and Simulates Inter-TX-Task Delays. 78
5.9 Signal Conversion from the Number of Transmit-Pending Messages to the Inputs for the Inter-TX-Task Delay Subsystem Blocks. 80
5.10 The Internal Implementation of the Inter-TX-Task Delay Subsystem Block. 80
5.11 The Internal Implementation of the Compute Ti Delay Subsystem Block. 83
5.12 Acquisition of Semaphore before Queuing Assessment 84

5.13 SimEvents Blocks that Implement the Transmit Queues, the TX Buffers and Simulation Data Logging mechanism. 85
5.14 The Queuing Assessment Mechanism. 86
5.15 The Transmit Queue (Queue 0) Mechanism. 87
5.16 The TX Buffer Mechanism. 90
5.17 The Internal Implementation of the 25ms TX Message Timing Management Subsystem Block. 91
5.18 The Internal Implementation of the Irregular Enqueue Detection Subsystem Block. 93
5.19 The Internal Implementation of the Enqueue Time Adjustment Subsystem Block. 94
5.20 The Internal Implementation of the Reserved Slot Logging Subsystem Block. 95
5.21 The Return Route for Successfully Transmitted Messages in Root Level of the SimEvents Model. 96
5.22 The Merge of Input Entity Routes before the Simulation Data Logging Mechanism. 98
5.23 The Simulation Data Logging Mechanism. 99
5.24 A Section of a Sample Model Generated Trace. 101
6.1 The Message IDs and Nominal Transmit Rates of Transmit Messages in Each Message Set, Listed in the Order of Internal Transmission Priority. 105
6.2 TX Task Arrivals During a Complete Hyper-Period in the 18-75-3 Simulation. 110
6.3 Expected Transmit-pending Messages in Each Instance of the TX Tasks. 111

6.4 Model Generated Trace of the the 18-75-3 Simulation under the Ideal Implementation, from the Start of the First Hyper-period, Part 1 . . 113
6.5 Model Generated Trace of the the 18-75-3 Simulation under the Ideal Implementation, from the Start of the First Hyper-period, Part 2 . . 114
6.6 Expected Transmit-pending Messages in Each Instance of the TX Tasks in the 19-50-5 Simulation. 118
6.7 Model Generated Trace of the 19-50-5 Simulation, Ideal Implementation, Part 1. 120
6.8 Model Generated Trace of the 19-50-5 Simulation, Ideal Implementation, Part 2. 121
6.9 Model Generated Trace of the 19-50-5 Simulation, Ideal Implementation, Part 3. 122
6.10 Model Generated Trace of the 19-50-5 Simulation, Ideal Implementation, Part 4. 123
6.11 Model Generated Trace of Ideal Implementation, the 19-50-7 Simulation, from the Start of the First Hyper-period, Part 1. 127
6.12 Model Generated Trace of Ideal Implementation, the 19-50-7 Simulation, from the Start of the First Hyper-period, Part 2. 128
6.13 Model Generated Trace for the 2^{nd} Instance of the 25ms TX Task. . . 129
6.14 Model Generated Trace for the 3^{rd} Instance of the 25ms TX Task. . . 130
6.15 The Enqueue Sequence of TX Tasks Arrived at Simulation Time 100ms. 131
6.16 The Actual Arrival of the 1^{st} Instance of Message 27. 132
6.17 The Enqueue Sequence of TX Tasks Arrived at Simulation Time 225ms. 133
6.18 The Actual Arrival of the 2^{nd} Instance of Message 27. 134

6.19 Autocorrelation of Normalized Regular Enqueue Times in Queue 0. . 138
6.20 Autocorrelation of Normalized Regular Enqueue Times in Queue 1. . 139
6.21 Autocorrelation of Normalized Irregular Enqueue Times in Queue 0. . 140
6.22 Candidate Probability Density Functions and the Histogram of Regular Enqueue Times in Queue 0. 141
6.23 Candidate Probability Density Functions versus Histogram of Regular Enqueue Times in Queue 1. 142
6.24 Candidate Probability Density Functions versus Histogram of Irregular Enqueue Times in Queue 0. 143
6.25 Autocorrelation of Normalized $t_{i,1}$. 144
6.26 Autocorrelation of Normalized $t_{i,2}$. 145
6.27 Candidate Probability Density Functions versus Histogram of $t_{i,1}$. . . 146
6.28 Candidate Probability Density Functions versus Histogram of $t_{i,2}$. . . 147
6.29 Autocorrelation of Normalized Measurement Jitter. 149
6.30 Candidate Probability Density Functions versus Histogram of Measurement Jitter. 150
6.31 The Data Collection Process for Message Transmit Rates Validation. 152
6.32 Message Transmit Rate Validation Statistics from a Single Simulation Run of the 18-75-3 Simulation Scenario. 154
6.33 Transmit Message Transmit Rate Validation Statistics from a Simulation Run of the 19-50-3 Simulation Scenario. 156
6.34 Another Sample of Transmit Rate Validation Result of the 18-75-3 Simulation Scenario. 158

6.35 Another Sample of Transmit Rate Validation Result of the 19-50-3 Simulation Scenario. 159
6.36 Intermediate Results: Numbers of Containment from All Simulation Runs of the 18-75-3 Simulation Scenario. 161
6.37 Transmit Message Transmit Rate Validation Overall Results: Percentage of Containment of the 95% Confidence Interval. 162
6.38 Creation of the List of Indices of TX Task Instances, in which the Message is Transmitted. 164
6.39 The Data Collection Process for Output Message Sequence Validation. 166
6.40 Output Message Sequence Validation Statistics from a Single Simulation Run of the 19-50-3 Simulation Scenario. 167
6.41 Output Message Sequence Validation Statistics from a Single Simulation Run of the 18-75-7 Simulation Scenario. 169
6.42 Determination of the Index of the First Mismatched Instance of the TX Task in Output Message Sequence Comparison. 170
6.43 Output Message Sequence Validation Simulation Statistics of All Simulation Runs for the 18-75-5 Simulation Scenario. 172
6.44 Overall Simulation Results for Output Message Sequence Validation. 173
6.45 Example of Discrepancies in Message Transmit Rates between Simulations Under The Ideal Implementation and The Simulated Host Environment. 176

Chapter 1

Introduction

The Controller Area Network (CAN) protocol was introduced over two decades ago and has gained world-wide popularity (Di Natale et al. (2012)). It has many desirable features such as low configuration and deployment costs, easy integration and simultaneous multi-ECU development, and boundable latency and utilization. The limited data rate (nominal rate of 500kb/s) has become its main drawback due to increasing bus load in modern vehicles, due to the addition of features such as user-facing features (such as the in-vehicle infotainment system and Advanced Driver-Assistance Systems), and diagnostic and security bus traffic. As a result, the overall bus utilization is reaching its limit and the transmission of messages has gradually become the bottleneck of using the CAN network. Careful design is essential to guarantee timely and reliable delivery of messages.

1.1 The problem

In industrial practice, Hardware test bench systems, for instance a hardware test bench available at a industrial partner consisting of a dual-core micro-controller unit (MCU) in an Electronic Control Unit (ECU) and Lauterbach hardware-in-the-loop (HIL) debugger, are built to thoroughly test the CAN-related functionalities of software such as the Dual FIFO CAN (DFC) stack. They are known for producing experiment results that are identical or at least considerably close to the outcome of the actual system. However, hardware test benches are typically difficult to set up, require a large amount of capital to build, and become a bottleneck for engineers late in the development life cycle since only one test scenario can be run at a time. Once built, they are incompatible with other micro-controller platforms and they tend to offer limited configuration options for only a fixed number of experimental scenarios. Each scenario takes considerable time to setup and run on the test bench.

There are also existing software simulation tools that are available for the CAN protocol. However, their implementations are highly abstracted and are based on invalid assumptions related to the details of the target system (Whinton (2016)), such as the number of transmit buffers (TX buffers) of a node on the CAN network, or that the host environment has no interference on the target CAN stack in real life, which leads to discrepancies in behaviors between the simulation and real life. On the other hand, these generally do not comprehensively possess all desirable features for simulations of modern CAN systems, more specifically, features such as flexible message queuing policy, configurable data logging options, support for dual-core control unit and so on. Analytical approaches can be useful, but existing analysis contains similar problems as the software simulation tools: unrealistic results compared to

actual system output due to a lack of system details. And if analyzing a relatively more complex CAN system, such as one with multiple transmitting nodes and large quantity of transmit messages, a dedicated tool needs to be development to handle the heavy load of computation.

1.2 Purpose

The purpose of this research is to develop a simulation model of the DFC stack, which overcomes the drawbacks of the hardware test bench systems while being able to produce simulation results identical or acceptably close the experiments on a hardware test bench and thus can be used to effectively analyze the behaviors of the the DFC stack.

The simulation model should offer the desirable features for simulations of modern CAN systems and these features should be implemented as configurable features that can be selectively enabled and configured with specific parameters to matched the target system. During simulations, the simulation model should be able to capture a large variety of internal data of the DFC stack and present the captured data in various forms, such as summary statistics or graphical plots. The simulation model should also be capable of simulating the loading effects of the host environment, which helps to provide more realistic simulation results that are identical or at least close to running the experiments of the same configurations on actual hardware. These simulation results can be further validated in terms of how well the host environment is modeled, so that, once validated, the simulation model can replace the hardware test bench as the testing environment, and thus reduce the need to invest in hardware test benches, resulting in savings in capital costs. Having immediate desktop access

to the simulation model also saves engineers time, reducing their requirements to book test bench time and go to the facility.

The simulation model is developed in the MathWorks MATLAB environment, which is a well-established industrial-standard platform that many other commercial simulation tools are built upon, such as Vector CANoe or dSpace tools. As a result, this would enable system-level simulations where the simulated CAN networks contain nodes that are implemented on different CAN platforms, such as adding the simulation model to an existing CAN network to analyze the impact on the CAN bus utilization and so on.

The simulation model can be useful not only when used alone, but also when used alongside with a test bench. As the number of test benches is often limited and configuration of a test bench is often difficult (and sometimes will even require the help of a trained personnel), the time an engineer can spend with the test bench can be very limited. If having the simulation model, the engineers can first conduct pre-experiment simulations to closely predict the outcomes of the test bench experiments in order to increase the efficiency of work while working on the test bench.

The implementation of the simulation model will focus on modeling the message transmission functionalities, more specifically the transmission of the periodic messages, due the limited time duration of this research and the fact that the point of interest in modern CAN simulations is the analysis of message throughput of the target CAN system in a potentially congested CAN network. Message reception of modern CAN nodes is not a major concern due to the processing capabilities of modern processing units (MCU, *etc*).

1.3 Contributions

First the thesis provides a high fidelity simulation model of a CAN stack that is widely used by an industrial partner. The simulation model is implemented in the industry-standard MATLAB environment, and it is easy to use and configure to model the hardware and software behavior of an ECU. Second the thesis provides a method to analyze patterns of behaviors in experiment data of the actual system, and to incorporate the models of the behaviors into the implementation of the simulation model to match the actual system, which consequently improves the fidelity of the simulations compared to the actual system. The model and method has been validated against a test bench at an industrial partner of this research. The simulation model is being adopted by production engineers at the industrial partner.

1.4 Organization of the Document

The remainder of the document is organized as follows: Chapter 2 covers the necessary background on the Controller Area Network protocol, related work, existing tools and the MATLAB/SimEvents environment, focusing on SimEvents features that will be used in the simulation. Chapter 3 describes the Dual FIFO CAN stack that will be simulated and Chapter 4 details the hardware test bench (host environment), experimental data and impact that the host environment introduces to the system output. The simulation model is covered by Chapter 5 and the simulation results and validation of the model are described in Chapter 6. Chapter 7 summarizes the results of the thesis and describes future work.

Chapter 2

Preliminaries

2.1 Controller Area Network (CAN)

The CAN 2.0 standard was introduced in the early 1990s by BOSCH GmbH. It defines the specification of the physical and data link layers (within the OSI model) of the CAN protocol.

2.1.1 Physical Components

The physical components for the CAN protocol contains the CAN bus, the CAN transceiver and the CAN controller.

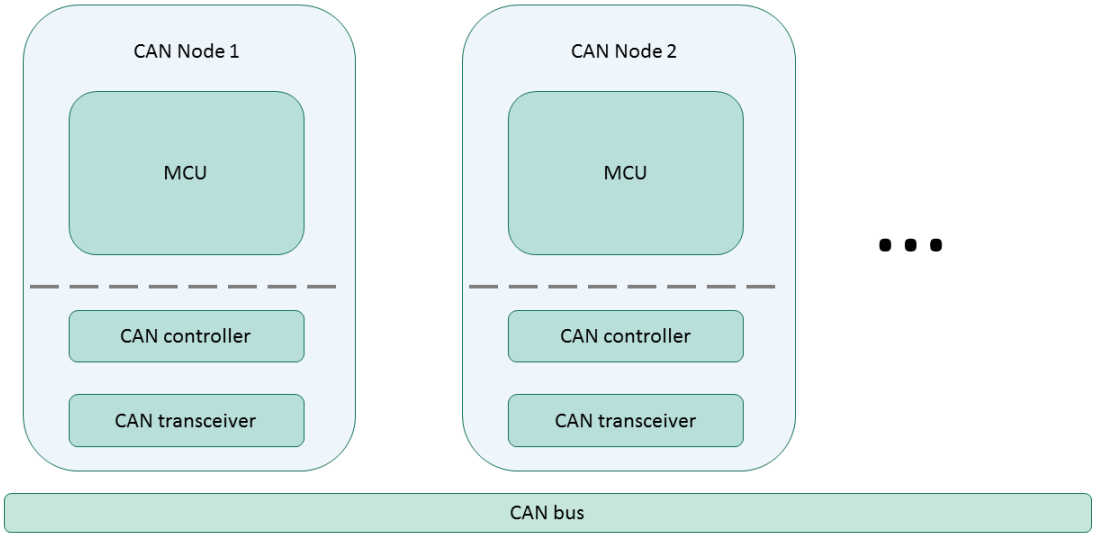

Figure 2.1: Typical Hardware Components of a CAN Network.

The CAN bus is a two-wire serial bus that terminates with a 120 ohm resistor at each end (2 wires labeled "CAN high" and "CAN low"). A "dominant" bit, logical 0, transmitted on the bus is represented by a voltage differential across the 2 wires of the CAN bus, while a "recessive" bit, logical 1, has no voltage differential. The CAN protocol allows multi-master serial communication, and each node (electronic control units, or ECUs, *etc.*) on the same network is capable of both transmission and reception. Two or more nodes are required on the bus for communication.

The CAN transceivers translate the data stream between the signal levels that are used by the CAN bus and the CAN controllers: converting from the CAN bus signal levels to CAN controller signal levels when receiving, and vise-versa when transmitting. The CAN controllers store received serial bits of CAN messages so that the processor (i.e. MCU of the receiving ECU) can fetch the received messages, and

send bit streams of out-going messages to the CAN transceiver when the processor commands to transmit.

Typically, ECUs are connected to the CAN bus through a combination of a CAN controller and a CAN transceiver (referred to as "CAN hardware"). The CAN hardware translates transmit messages from digital bit streams to CAN waveforms when transmitting, and captures waveforms of received messages from the bus and translates them back into digital bit streams. The transmit messages need to be loaded into transmit buffers (or "TX buffer") in the CAN controller so that the CAN controller can start the transmitting process as data frames on the bus. Received messages are either stored in receive buffers ("RX buffer") or discarded based on RX buffer policies (i.e. message ID filtering, message-ID-to-RX-buffer mapping, *etc.*).

2.1.2 Data Format

The format of CAN message frames are defined in the data link layer. There are several different types of message frames that can be transmitted on the CAN network. The Data frames carry data from the transmitting node while remote frames request data to be sent by a certain node. The Error frames and the Overload frames can be sent by the nodes that detect error or overload conditions. For the purpose of this research, only data frames are considered.

Data frames contain the following fields:

- Start of Frame (SOF)
- Arbitration (message ID)
- Control (DLC)

- Data
- Cyclic Redundancy Check (CRC)
- Acknowledgment (ACK)
- End of Frame (EOF)

Data frames can be in either standard frame format or extended frame format. In standard frame format, only 11 bits of message ID is stored in the arbitration field, while the extended frame format allows a message ID of 29 bits in the arbitration field. The data field can be from 1 to 8 bytes depending on the data length specified in the control (DLC) field. The SOF field is a dominant bit and it donates the start of frame transmission. The CRC field is a 15-bit cyclic redundancy check followed by 1 recessive bit of delimiter. The ACK field consists of a 1-bit acknowledgment signal, which is transmitted recessive by the transmitting node but is written dominant by any receiver node that has successfully received the message, followed by a 1 recessive bit delimiter. The EOF field is 7 recessive bits. The SOF, control, CRC, ACK, and EOF fields have the same length for all data frames.

2.1.3 Arbitration

The CAN protocol uses a lossless bit-wise arbitration scheme for contention resolution in its data transmission, which requires all the nodes on the same network to be synchronized to sample every bit on the bus at the same time. If a node transmits a dominant bit while another node transmits a recessive bit at the same time, then there is a collision on the bus and the dominant bit wins the arbitration. The result is a dominant bit, or logical 0, shown on the bus. When multiple nodes attempt to

transmit on the bus at the same time, if a node reads a dominant bit when it transmits a recessive bit, the node will notice the collision and that it has lost the arbitration. The node will then stop the transmission of its message and attempt to re-transmit that message at the start of the next frame on the bus. As a result, the message that has won the arbitration will go through on the bus without any delay. This scheme allows the assignment of priorities to messages on a CAN network through the assignment of their message IDs. An example is shown in Figure 2.2, which shows the arbitration of the 8 bits of the message IDs between messages transmitted by 3 ECUs as well as the resultant data stream on the bus.

	BIT							
	1	2	3	4	5	6	7	8
ECU 1	0	0	0	1	1	0	1	1
ECU 2	0	0	1	-	-	-	-	-
ECU 3	0	0	0	1	1	1	-	-
BUS	0	0	0	1	1	0	1	1

Figure 2.2: An Example of Arbitration between Three Messages.

The bit streams of the message IDs in binary are shown from the most significant digit to the least significant digit from the left to the right, which is also the order of which the message IDs are transmitted in the data frames. ECU 2 loses arbitration at Bit 3 and stops transmitting as the ID of its message has a recessive bit while the other two messages have a dominant bit. ECU 3 later loses arbitration to ECU 1 at Bit 6 and stop transmitting. The message transmitted by ECU 1 wins the arbitration as its message ID has the smallest numerical value. In other words, the smaller the numerical value of a message ID is, the higher the transmit priority the message will have.

Each message ID should be unique in each CAN network, otherwise if two nodes transmit the same message at the same time, they would both keep transmitting after the arbitration field (ID) and cause an error.

2.1.4 Bit Stuffing

CAN uses non-return to zero (NRZ) coding, which is represented by the amount of potential difference between the CAN high and CAN low wires. As CAN requires all nodes on the network to synchronize to every single bit on the bus, it is necessary to introduce a scheme to provide enough transitions to keep the nodes synchronized. Thus during transmission, a single bit of opposite polarity is inserted after 5 consecutive bits of the same polarity. This is called bit stuffing. The stuffed data frames are de-stuffed by receiving nodes. An example is shown in Figure 2.3. In the figure, (a) is the original bit stream from the transmitting node. During transmission, bit stream (b) is presented on the bus due to bit stuffing. Bit stream (c) is observed at receiving nodes after the de-stuffing, which is identical to the original bit stream from the transmitting node.

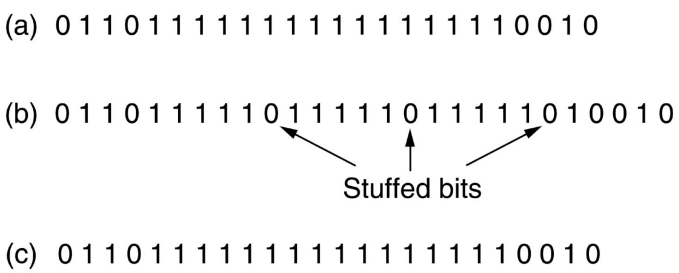

Figure 2.3: An Example of Bit Stuffing on CAN data stream.

2.2 Related Work

Hofstee and Goense (1999) presents an early example of CAN simulation, which is designed for a very specific CAN application for agricultural tractors. The focus of the simulations is the priorities of the message, which bus certain ECUs are connected to and the traffic levels of a specific bus. The main goal of the research is to use simulation to validate that the tractor CAN system in compliance with the ISO standard for CAN. The simulation does not concern itself with the internal implementation of the ECUs and how the implementation details would affect the timing performance of the ECU such as the message transmit rates.

Prodanov et al. (2009) discusses simulation of CAN transceivers at the signal level. The research focuses on the electrical characteristics of the system in order to evaluate the corner cases of electromagnetic interferences or signal integrity in CAN systems, and to examine different types of fault scenarios such as short-circuits. In contrast, the research of this thesis models communications in the higher layers of the OSI model, and the focus is the queuing behaviors, correctness of output message sequences and the transmit rates of messages.

There are several other works on CAN simulation that are more similar to the research of this thesis in terms of the focus on timing performances and the level of details for ECUs on a CAN network. Hao et al. (2011) discusses a simulation that specifies a single non-preemptive TX buffer, however it does not offer any level of configurability in its model of ECU such as more than one TX buffer or scheduling behavior. This is very restrictive and is not the case in most ECUs. The simulations described in Herpel et al. (2009) and Matsumura et al. (2013) offer some network configuration options to generate system level models for simulation. However, similar

to Hao et al. (2011), these simulations are also limit to a single non-preemptive TX buffer in their ECU models, and provide no configurability for TX buffers or other details of the ECU.

2.3 Existing Tools

There are several tools for CAN simulations. One of the popular simulation tools currently in use is RTaW-Sim by Real Time at Work. According to its reference manual, the tool includes a variety of features and functionalities such as a variable number of TX buffers, configurable TX task periods, software queuing by FIFO or priority, *etc* (Matsumura et al. (2013)).

CANoe CANalyzer simulation tool is a popular commercial CAN network design tool by Vector Informatik GmbH. It offers several configuration options for the implementation details of the ECU by default, and rest of configurable items are mostly the message database of the CAN network or system description (Vector Informatik GmbH (2014)). Since only a limited number of Vector and OEM model libraries for ECUs are included by default, changing the configuration and behaviors of the ECU model will require a custom-built library, which makes the modification of the configurations rather cumbersome.

Another simulation tool developed at Nagoya university, Japan, called *A Simulation Model of Controller Area Network (CAN) for OMNeT++*, offers several but not all required features for the research of this thesis. The simulation tool features a partial priority queue, where messages that have already queued to transmit on the bus are not to be preempted from their queue position, which is different from the non-preemptive FIFO queues used in this research. Furthermore, the rates of the TX

tasks are not configurable.

There are existing CAN simulation tools developed in the MathWorks SimEvents and Simulink platform, which provide an environment for developing discrete event simulation tools. TrueTime is a tool developed in Simulink using the Control System Toolbox for simulation of networked and embedded control systems (Cervin et al. (2010)). Again, it lacks features such as configurable TX task periods and FIFO transmit queues. MathWorks also provides a demo CAN model, *Effects of Communication Delays on an ABS Control System*, which is connected to an Anti-lock Braking System (ABS) controller model (MathWorks (2014)). The demo model does not concern non-preemptive buffers and FIFO transmit queues, and the arbitration of messages is dependent on the connected transmit input port of the bus model rather than the message priority. Consequently, the demo model is only useful as a concept of how CAN systems may be represented in the SimEvents environment rather than a CAN simulation model, and is not suitable for the research in this thesis.

The most comprehensive CAN simulation tool seems to be the Vector CAN simulation tool introduced in Whinton (2016), the "High-Fidelity Vector CAN Stack Simulation Model", which was developed at McMaster University, Hamilton, Canada. The tool is implemented in the SimEvents environment (Version 2014), and provides following features:

- Configurable quantities of transmit (TX) and receive (RX) buffers.
- Message to (TX or RX) buffer mapping.
- Configurable buffer-loading policy (polling versus interrupt).
- Configurable TX Task and buffer loading polling rate.

- ECU clock drift and initialization time.
- Simulated CAN bus with configurable baud rate, error rate and bit stuffing options.
- Multiple-ECU CAN network simulation, and/or multiple CAN network connections for individual ECU.

The Vector CAN simulation tool developed in Whinton (2016) consists of two parts: a custom SimEvents library and a set of model generation scripts. The SimEvents library contains two custom subsystem blocks (or, models), the generic ECU block and the generic CAN bus block. The model generation scripts are written in MATLAB code (saved in ".m" files) and are used to parse a system description and then assemble the system-level model with replicated instances of the ECU and bus blocks. The system description is a combination of an ARXML file and a custom Excel spreadsheet, which provides parameters of the target CAN network including the TX and RX message information and the message ID to (TX and/or RX) buffer mapping of each ECU in the simulation, interconnections between the ECUs and CAN networks, *etc.*, or in other words, parameters for all the configurable features listed earlier. The parameters extracted from the system description are used to configure their corresponding ECU and bus blocks when the system-level model is assembled. In summary, the Vector CAN simulation tool by Whinton (2016) contains all the features that are offered by other existing simulation tools described above and also allows easier modification of the configuration of ECUs, CAN bus and CAN network layout. However, similar to other tools, it utilizes a different queuing mechanism than the target system of this research.

Common to all the existing tools introduced above, the reserved slot mechanism in the FIFO transmit queues of the DFC stack is not addressed in their implementations. Furthermore, these tools ignore the details of the host environment of the simulated CAN system and any possible impact that the host environment may have on the behavior of the CAN system, which, as later discussed in detail in Chapter 4, can have significant effect on the output of the CAN subsystem in the cases studied for this research.

2.4 MathWorks Software

2.4.1 MATLAB

MATLAB (MATrix LABoratory), Simulink, and SimEvents are all software tools designed for technical computation in engineering and the sciences. They are all part of the MATLAB suite of tools published by MathWorks. MATLAB is built on a variety of libraries including C, C++, and Java, and provides an interpreter for its own scripting language (MATLAB functions or script saved in ".m" file format). Besides allowing users to perform various types of calculations, MATLAB provides functionalities for generating various types of graphical plots (2D or 3D plots, *etc.*). The most distinctive feature of the MATLAB environment is that it uses matrices as base data types and provides robust matrix manipulations and vectorized operations with easy graphing and data visualization capabilities.

2.4.2 Simulink

The Simulink environment is the graphical modeling language designed for ease-of-you and rapid development of models. Although developing in Simulink is quite different from coding in MATLAB, Simulink allows incorporating MATLAB code directly in its models, and provides an API for model generation, modification and analysis in MATLAB scripts. Fundamental block elements form the base syntax of the Simulink language, which define the output signal as a function of its input signals and/or parameters. The fundamental blocks include things like constants, addition and multiplication operations, integral and derivative operations, and signal generation. Instances of these blocks may be declared, configured and interconnected in a model to define new and more complex models such as analogue filters, PID controllers, *etc.* Other blocks implement control flow behaviors, such as "if" conditions or signal switches, which allow for mode switching or decision making. Signal scopes are also provided in the Simulink environment to allow signal levels to be displayed at real time or to be saved in MATLAB workspace.

2.4.3 SimEvents

SimEvents introduces the concept of discrete event simulation to Simulink. In contrast to Simulink, where signals are continuous and always have a value, the SimEvents signals are discrete and do not always have a value. Other than this, signals are the same and can be passed through between blocks in both environments, except that when continuous signals need to be sampled at the appropriate event occurrences in order to be converted to discrete signals, or discrete signals need to hold their most recent sample values in order to be converted to continuous signals. The SimEvents

environment also introduces the concept of entities, which are discrete objects that propagate through the entity ports of SimEvents blocks in a model. They can be used to model data packages in communication protocols or partial products in an assembly process. Entities can have attributes attached to them, which are key-value pairs that can be written or read by certain SimEvents blocks, and can be used to store values of signals. Since entities are discrete objects, they may experience queuing, blocking, service times and other sources of time delays that are implemented by the fundamental blocks in SimEvents. There are also several entity routing and management blocks available to control the flow of entities such as switching entities into one of several output paths from a single input paths based on an input signal or the value of a specific attribute attached to the input entities, *etc.*

The simulation model discussed in this thesis has been developed in MathWorks SimEvents environment (Version 2015a) primarily because it is a comprehensive modeling and simulation environment that is well suited for modeling of event-based systems and communication protocols such as CAN. An example of a simplified CAN models in the SimEvents is provided by MathWorks. It can be used as a basis for many behavioral elements required in the higher-fidelity models.

Developing a simulation model in the same platform as the The Vector CAN simulation tool developed in Whinton (2016) allows for possible future work such as system-level simulation of a CAN network containing both Vector CAN stack ECUs and the DFC stack ECUs, which opens up possibilities in analysis of more complicated CAN networks. For this purpose, the DFC stack simulation model is implemented so that it can utilize the generic bus block in the Vector CAN simulation tool, and thus is able to transmit and receiver messages on a network connected to the generic bus

block.

Chapter 3

The Dual FIFO CAN Stack

As the behaviors of the DFC stack are determined by both the internal implementation of the DFC stack and characteristics of the host environment (hardware test bench with a dual-core ECU and the RTOS, in the case of this thesis), the complete introduction of the DFC stack will be presented in 2 separate chapters, this chapter and Chapter 4. This chapter will cover the specifications of the DFC stack, providing details about the internal implementation, and Section 4 will discuss the host environment, experiment data and the impact of host environment on the output of the DFC stack.

3.1 Overview of Dual FIFO Queue CAN Stack

The DFC stack is part of the RTOS that runs on the dual-core ECU of the hardware test bench, and it provides the CAN related functionalities to its upper layer applications in the RTOS. It implements 2 software abstraction layers: the CAN Handler layer and the CAN Device Driver layer, on top of the CAN hardware layer and the

CAN bus layer (both physical hardware), as demonstrated in Figure 3.1. The DFC stack can be customized by several configurable calibration values, which determine the behavior of the DFC stack, such as queuing policy, *etc.* The user can also arbitrarily define the transmit messages of the DFC stack, including the message IDs, DLCs, message periods and internal transmission priority. The calibration values and the transmit messages are defined offline before the DFC stack is initialized and can not be changed during run-time.

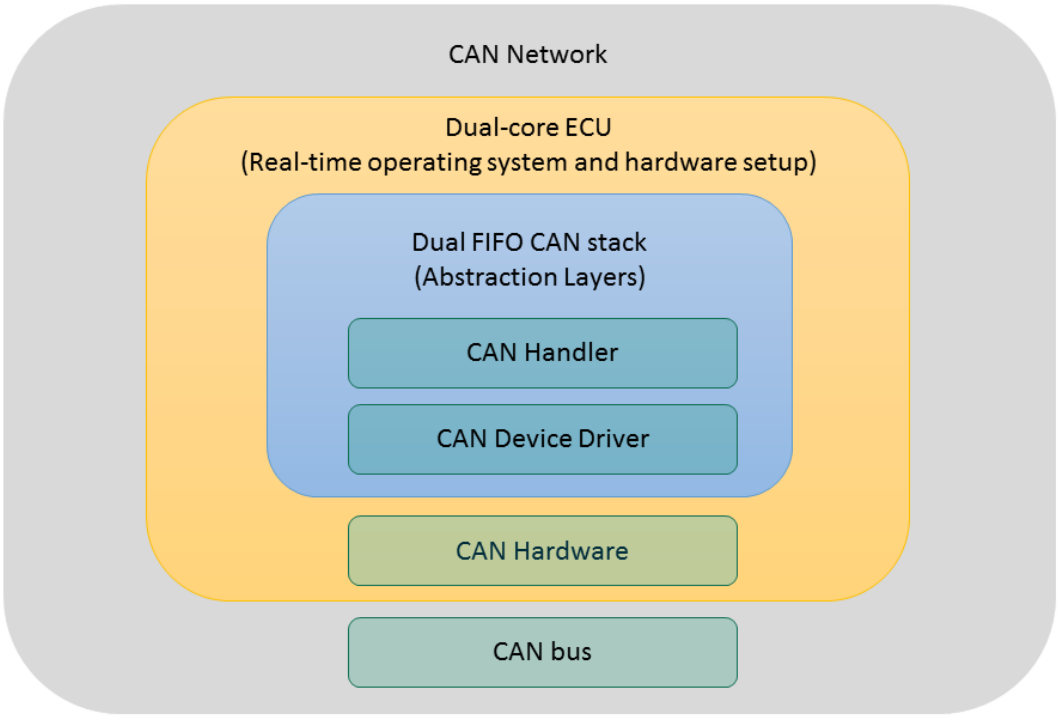

Figure 3.1: Dual FIFO Queue CAN Stack Abstraction Layers

3.1.1 CAN Handler

The CAN Handler provides an interface for upper layer applications in the RTOS, which enables the applications to communicate serial data on the CAN network without concerns about the implementation of the serial data network.

The CAN Handler implements three periodic TX tasks, each of which manages the transmission of a subset of the periodic transmit messages of the DFC stack, as shown in Figure 3.2. The TX tasks execute once every 6.25ms, 12.5ms and 25ms, respectively, which are frequently used loop rates of TX tasks for the CAN protocol in education (Di Natale (2009)), research (Di Natale et al. (2012)) and industrial practice (Jiang and Zhang (2016)). The CAN Handler utilizes the routines provided by the CAN Device Driver that build and transmit the messages on the CAN network through the lower layers. More details will be provided in Section3.2.

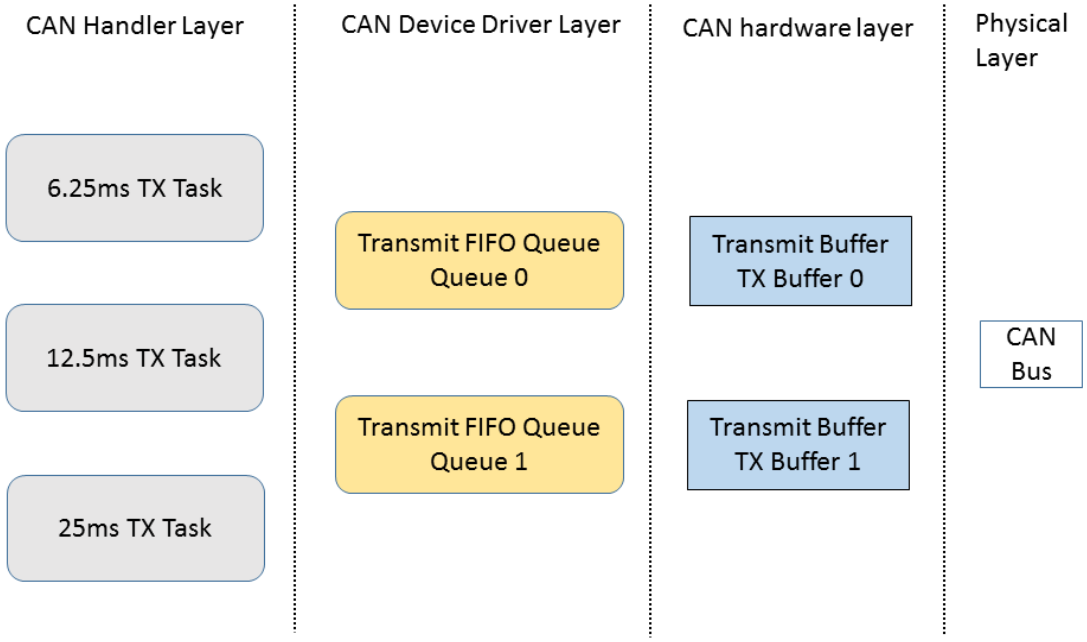

Figure 3.2: Components in the Layers of Dual FIFO CAN stack.

3.1.2 CAN Device Driver

The CAN Device Driver connects the CAN Handler and the CAN hardware, providing an interface that facilitates the exchange of CAN messages between the two layers while hiding the specifics of the CAN hardware. Based on the actual hardware setup of the CAN node, it is possible that the CAN Device Driver needs to be connected to multiple CAN controllers in order to provide sufficient resources to the higher layers.

For each CAN network that the DFC stack is connected to, the CAN Device Driver implements two non-preemptive FIFO (first-in-first-out) queues (Queue 0 and Queue 1) for the transmit messages. The two FIFO queues are implemented as circular buffer structures. Queue 0 is intended for transmission of messages with relatively higher priorities, and the rest of the messages are to be transmitted through Queue 1. A designated TX buffer is assigned to each of the FIFO queues from the CAN hardware layer by the CAN Device driver (TX buffer 0 for Queue 0 and TX buffer 1 for Queue 1). Messages intended of one of the FIFO queues are transmitted on the bus only through the designated TX buffer of that queue, and are not allowed to use the buffer of the other queue. The buffer usage is relatively restricted in the DFC stack in contrast to other popular platforms, such as the STMicroelectornics' STM32F4 micro-controller series, which have multiple TX buffers, are capable of transmitting a message through an arbitrarily selected buffer, and support various buffer transmit priority assignments when more than 1 TX buffers are loaded with messages (STMicroelectronics (2017)).

3.2 Message Transmission

The basic flow of transmission of a message in DFC stack is shown in Figure 3.3. Once a message becomes pending for transmission during the execution of its TX task, the TX task will attempt to transmit the message. The eligibility for transmission of a transmit-pending message is decided through the queuing assessment, which decides whether the message can be transmitted during the current instance of the TX task. The queuing assessment determines the eligibility based on the configurable calibration values, transmit rate of the assessed message and several real-time factors (queue depth and TX buffer availability of the target FIFO queue). If a message is eligible for transmission, it can be transmitted in 2 different paths: (i) if the target FIFO queue is empty and the corresponding TX buffer is available, the message will bypass the queue and enter the TX buffer directly for transmission on the CAN bus, and (ii) if the TX buffer is occupied but the message is eligible for entering the target queue, the message will be placed into the queue and waits for its turn to be loaded into the TX buffer. After the queuing assessment, a transmission-eligible message is then built and loaded into the target queue (or TX buffer directly) by the CAN Device Driver routines, which are invoked by the TX task of that message. For a message that is not eligible for transmission, it will be assessed again in a later instance of its TX task. Detailed discussion is provided in the following subsections.

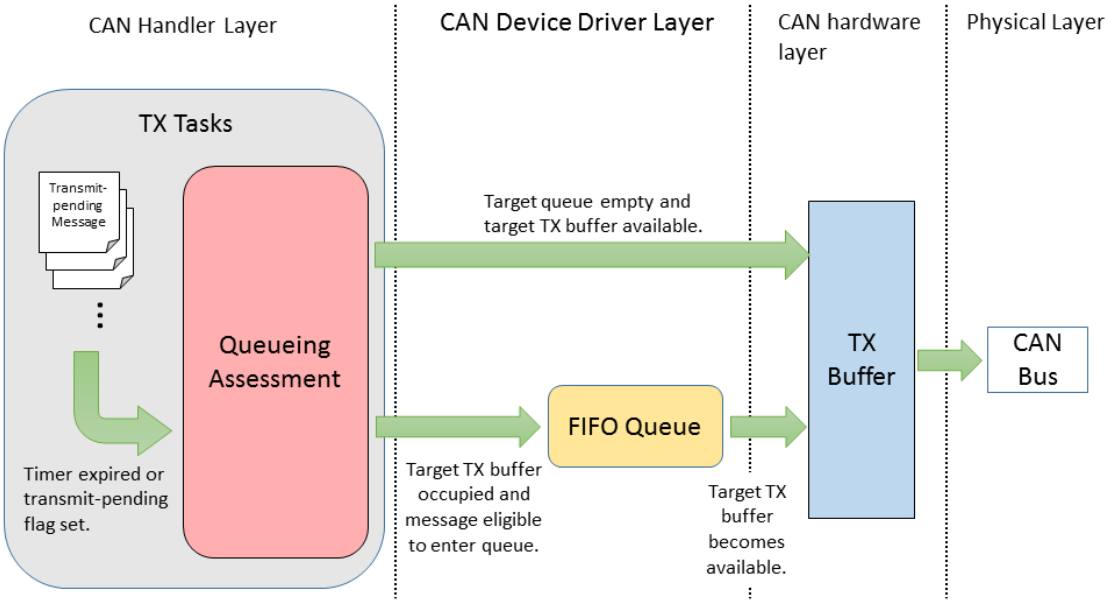

Figure 3.3: The Basic Flow of a Transmit Message in the DFC Stack.

3.2.1 Message Enqueuing

The CAN Handler implements three periodic TX tasks that have periods of 6.25ms, 12.5ms, and 25ms, respectively. The TX tasks manage the periodic transmit messages of the DFC stack, which are divided into 3 subsets based on their transmit rates and are assigned to these TX tasks. Since the loop rates of the slower TX tasks are integer multiples of the more frequent TX task, there are scenarios where more than one of them arrive at the same time. In such cases, the order of execution of these TX tasks is from the most frequent TX task to the least frequent TX task.

The CAN handler keeps a transmit-pending flag for each of the periodic messages, indicating if the corresponding message is pending for transmission, or in other words, needs to be assessed for queuing. Messages with transmit rates less than or equal to 6.25ms are managed by the 6.25ms TX task, and for this reason, the 6.25ms TX

task sets the transmit-pending flag of all its messages in every execution. Similarly, the 12.5ms TX task manages messages with transmit rates smaller than or equal to 12.5ms but greater than 6.25ms, and its messages are pending for transmission in every execution. However for the 25ms TX task, which manages messages with transmit rates greater than 12.5ms, there may be messages that have transmit rates greater than 25ms. As a result, there is a timer associated with each of the 25ms TX task messages to keep track of these slower messages that need to be transmitted only once every several instances of the 25ms TX task. When the 25ms TX task executes, it first updates the timers of its messages, then sets the transmit-pending flag of any message that has an expired timer. Once the a message become pending for transmission, its transmit-pending flag will stay set until it is either placed into a FIFO queue or loaded into a TX buffer directly for transmission on the bus.

All periodic message from the 6.25ms TX task are to be transmitted only through Queue 0, which is intended for messages with faster transmit rates. The 12.5ms TX task messages can be transmitted through either Queue 0 or Queue 1, depending on the transmit queue threshold calibration (*DFC_Queue0_TxThresh*). If this calibration value is set to 12.5ms or greater, then *all* the 12.5ms TX task messages will be transmitted through Queue 0, otherwise through Queue 1. As for the messages that are managed by the 25ms TX task, the target FIFO queue is decided individually for each message, by comparing the transmit rate of each 25ms TX task message to the calibration value *DFC_Queue0_TxThresh*. If the transmit rate of a message is smaller than *DFC_Queue0_TxThresh*, then the message is to be transmitted through Queue 0. Otherwise, the message is to be transmitted through Queue 1.

When the CAN Handler is initialized, the subset of periodic messages for each

TX task is provided in the form of an array, listing the messages in their internal transmission priorities. The internal transmission priority determines the order in which a TX task transmits its messages, in other words, the order for setting the transmit pending flag (and updating the timer in the 25ms TX task) and for the queuing assessment during the execution of a TX task. More details of internal transmission priority is provided in Section 4.1.2.

The FIFO queues managed by the CAN Device Driver are non-preemptive circular buffer structures. The total number of slots in each queue can be individually configured by the user, as long as it is above the minimum requirements: minimum of 10 slots in Queue 0 and minimum of 20 slots in Queue 1. Each slot in a queue is labeled with a slot number ranging from 1 to the maximum number of slots. Each FIFO queue is associated with a non-preemptive TX buffer. TX buffer 0 only accepts messages from Queue 0, and TX buffer 1 only from Queue 1. Once a message is loaded into a TX buffer, the buffer will start the transmission of the CAN frame on the bus immediately.

For messages that are loaded into the FIFO queues, they are placed in one of the two types of slots, the reserved slots and the non-reserved slots. The reserved slot mechanism is implemented according to the method specified in Kaufer et al. (2015). The calibration value *DFC_TxNotReservedQueueX*, where X is the queue number, can be customized by the user to specify the number of non-reserved slots in a queue. *DFC_TxNotReservedQueueX* has to be less than or equal to the maximum number of slots of the queue, which means that it is possible for a queue to consist of solely non-reserved slots if the calibration is equal to the total number of slots. The remaining slots aside from the non-reserved slots are the reserved slots. During the queuing

assessment of a message that is not to directly enter a TX buffer, the message is eligible to enter the target queue if (i) there is at least one available non-reserved slot, or (ii) in the case where all the non-reserved slots in that queue have been occupied but at least one reserved slot is available, if the transmit rate of that message is less than or equal to the transmit rate threshold for reserved slots (configurable calibration value *DFC_TxReserveThreshQueueX*, where X is the queue number) of the queue. In any other scenarios, such as a full queue or transmit rate exceeding the threshold, the message will not be eligible to enter the queue, and will be delayed and assessed again during the execution of the next instance of its TX task. The reserved-slot mechanism aims to prevent scenarios where a more important message is pending for transmission but the queue is filled with less important messages, which is very likely to delay the transmission of that message to a later instance of its TX task and possibly causes the message to miss its deadline. By having the reserved-slot mechanism, the depth of queues is always kept below a certain level to reserve room for more important messages. Due to the nature of the circular buffer structure, the reserved slots are not fixed to any particular slots in the queue, as the head and the tail of occupied slots are constantly moving around the circular buffer.

When a message leaves a queue, the slot released by that message will be counted as a reserved slot if the number of currently occupied reserved slots is less than the total number of reserved slots. When all the specified number of reserved slots are available in a queue, newly released slots will be counted as the non-reserved slots. By doing this, newly released slots at a busy time of message transmission will always be first reserved for more frequent messages that are eligible to enter the reserved slots.

3.2.2 Message Dequeuing

Message dequeuing is performed during the execution of the transmission-complete interrupt service routine (ISR), which is implemented and executed by the CAN Device Driver. The transmission-complete ISR is responsible for moving messages from the FIFO queues to the corresponding TX buffers, and is triggered by transmission-complete interrupts from the TX buffers. Whenever a TX buffer successfully transmits a message on the bus and becomes available, it generates a transmission-complete interrupt to inform the CAN Device Driver to execute the ISR. The transmission-complete ISR then checks if there is any message waiting in the corresponding FIFO. If there are messages still queued for transmission, the transmission-complete ISR moves a message from the head of the queue to the TX buffer. If the queue is empty, the transmission-complete ISR exits with no further action.

While the dequeuing of messages is interrupt-based, it is typically initialized by a message that bypasses the FIFO queue and directly enters the TX buffer, and this message tends to be the first message of a TX task. When a TX task assesses its first pending message, its more likely that the target FIFO queue and its corresponding TX buffer are empty compared for the rest of the messages, and consequently the first message of a TX task is more likely to directly enter the TX buffer. Once this message is transmitted on the bus and the TX buffer empties, a series of interrupt-based dequeuing is started for messages that have been placed into the queue. However, it is also possible for the first message of a TX task to be loaded into a FIFO queue, if there are still messages left in the FIFO queue and/or in the TX buffer from a previous TX task execution.

Chapter 4

Hardware Test Bench and Experiment Data

In order to validate the DFC stack simulation model, real-life data of the DFC stack has been collected from experiments on the dual-core ECU hardware test bench. This chapter will discuss the details of the hardware test bench, the experiments with the DFC stack, and the knowledge about the DFC stack and the host environment (RTOS and hardware test bench) gained through the observations of the experiment results.

4.1 The Hardware Test Bench

The test bench consists of a CAN network that has a single ECU, which has generic dual-core MCU connected to a dSPACE hardware-in-the-loop (HIL) simulator. A Lauterbach hardware debugger is attached to the ECU to record data. The DFC stack is executed in a RTOS that runs on the ECU. Due to the limited time duration of this research, experiments conducted on the test bench consist of only a small

number scenarios where the ECU of the DFC stack is the sole transmitting node on the CAN network, in order to reflect the behavior of the DFC stack without any interference of bus load from other nodes on the same network.

4.1.1 The DFC stack and Operating System Scheduling

The tasks inside the DFC stack are separately executed on both cores of the processor, as shown in Figure 4.1. While one of the cores is used for enqueuing of the messages (execution of the TX tasks in CAN Handler as well as the enqueue routine provided CAN Device Driver that builds and loads messages into the queues), the other core is for dequeuing of the messages (execution of the transmission-complete ISR that removes messages from the queues and loads them into the TX buffers). In other words, both cores need to access the contents of the FIFO queues and TX buffers from time to time. In order to prevent simultaneous access to the queues and buffers between the enqueue and dequeue related routines, a semaphore is used to enforce mutual exclusion of the critical sections of the routines. For the tasks performed by the enqueue core, the critical section is the queuing assessment of messages, during which the enqueue core reads reads and modifies the queue depth. For the dequeue core, the critical section is when the transmission-complete ISR removes a message and loads it into a TX buffer. This is essential for the DFC stack to function correctly, since during the queuing assessment the depth of the queue is an important criterion and any simultaneous dequeue action would stop the TX task from obtaining the up-to-date information.

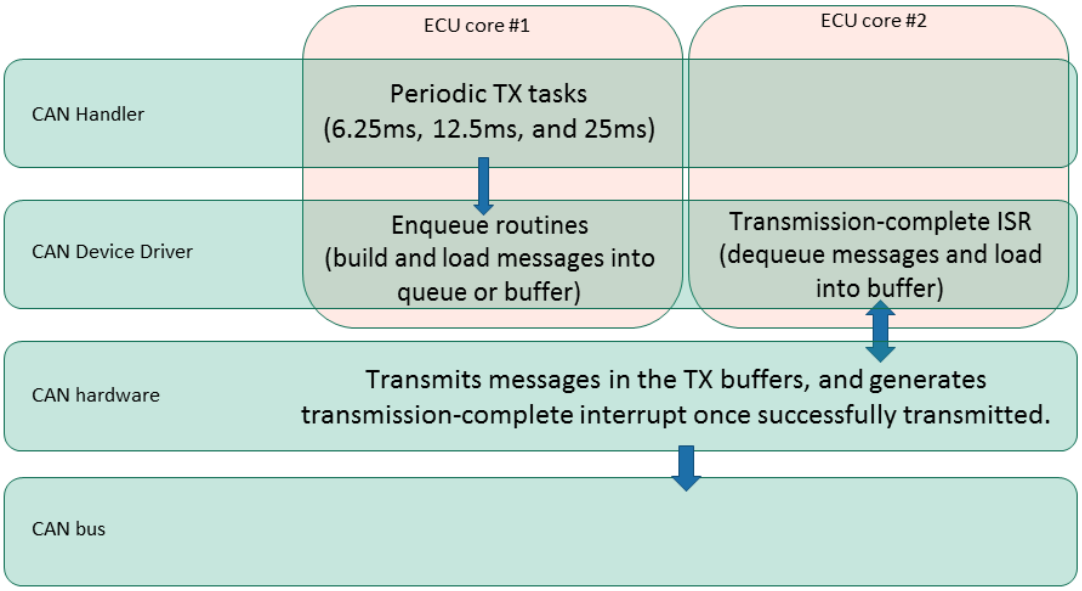

Figure 4.1: The DFC Stack and Dual Core ECU

Due to the fact that the DFC stack is a component of the RTOS, there are other tasks running in the RTOS that may effect the timing of the tasks and ISR in the DFC stack. For instance, some of the tasks in the RTOS may have higher priority and it is possible that their executions will delay the TX tasks of the DFC stack and cause jitters in the invocation or execution times of the TX tasks.

4.1.2 Internal transmit priority of TX Messages

The internal transmit priority of the transmit messages of the DFC stack can be configured through the Bus Number As the ECU of the test bench is capable of communicating on multiple CAN networks simultaneously, the transmit messages in the DFC stack can be configured to be transmitted on different buses (Bus A, Bus B, *etc.*) even for messages managed by the same TX task. The transmit messages

are assigned transmit priorities internal to the DFC stack based on their bus number. The internal transmit priority decides the order in which the messages are assessed for queuing and transmission. For instance, if a number of messages are pending for transmission during a TX task invocation, the TX task will first attempt to transmit messages that are labeled Bus A, and then messages labeled Bus B, and so on. As a result, depending on the configuration of the message set of the experiment, the internal transmit priorities of these messages may differ from the priorities on a regular CAN network (based on numerical value of message ID: the smaller the numerical of the ID, the higher the priority on the CAN bus). During the test bench experiments, the transmit messages are assigned with different bus numbers, however they are transmitted on the same and only physical CAN bus that the ECU is connected to. The resultant internal transmit priority is shown in Figure 4.3. The message IDs are listed in descending orders of transmit priorities from top to the bottom in the message ID columns.

As the host environment of the DFC stack may use any customized internal transmit priority schemes, it is crucial that the simulation model can be configured to flexibly use the same scheme as the host environment in order to produce the same output sequence of the transmit messages.

4.1.3 Experiment Scenarios and Captured Experiment Data

Due to limited access to the hardware test bench and the time consumption for configuring the test bench and debugger, it is possible to only conduct a small number of experiments. The test bench experiments consist of 2 configurations of the DFC stack, and 7 sets of transmit messages are tested on each configuration, which yields a

total of 14 scenarios. A single experiment run has been conducted for each experiment scenario, and the debugger has captured events for a duration of 1 second in each experiment.

The 2 configurations of the DFC stack differ in the number of reserved slots in Queue 1 and the message transmit rate threshold for entering those reserved slots, as shown in Figure 4.2, and the rest of the calibration values are identical throughout the experiments, which are the transmit rate threshold for entering Queue 0 (*DFC_Queue0_TxThresh*), the total number of slots in the FIFO queues and the number of reserved slots in Queue 0 (*DFC_TxNotReservedQueue0*).

Calibration value	Configuration 1	Configuration 2
DFC_Queue0_TxThresh	6.25ms	
Totoal number of slots in Queue 0	10	
DFC_TxNotReservedQueue0	10	
Totoal number of slots in Queue 1	20	
DFC_TxNotReservedQueue1	18	19
DFC_TxReserveThrshQueue1	75ms	50ms

Figure 4.2: Calibration Values in the 2 Experiment Configurations of the DFC Stack.

The calibration *DFC_Queue0_TxThresh* is set to 6.25ms, which means that only messages managed by the 6.25ms TX task are to be transmitted through Queue 0 and messages managed by the 12.5ms and 25ms TX tasks are to be transmitted through Queue 1. The value 6.25ms is chosen because There is a total of 10 slots in Queue 0 and a total of 20 slots in Queue 1. The calibration *DFC_TxNotReservedQueue0* is

set to 10, which means there is no reserved slots in Queue 0. For the differences between the two configurations, while one configuration specifies 18 non-reserved slots in Queue 1 and that only messages with transmit rate faster than 75ms can enter the reserved slots, the other configuration specifies 19 non-reserved slots in Queue 1 and an entering threshold of 50ms. Due to limited access to the test bench and the fact that the only way to setup the test bench experiment in order to observe different behaviors of the reserved-slot mechanism is to modify the reserved-slot mechanism related calibration values (in contrast, it is possible to use different sets of transmit messages to analyzed the behaviors determined by the queue capacity related calibration values), the number of reserved slots and the entering threshold of Queue 1 are chosen to be the variables to reflect differences in the simulation results (it is sufficient to observe the reserved-slot mechanism in only Queue 1 because the reserved-slot mechanism behaves under the same logic in both queues, and that no differences in Queue 0 could also avoid potential uncertainty when having more variables). Between the 2 configurations of the reserved slots in Queue 1, with less reserved slots, the transmit rate threshold of entering the reserved slots should be set to a smaller value (faster transmit rate) to ensure that those slots are reserved for only the even more frequent messages. As for the identical calibrations between the configurations, their values are chosen from one of the several commonly used values in industrial practice.

For each of the two configurations, 7 different sets of transmit message are used to conduct experiments. Figure 4.3 shows the IDs and periods of the transmit messages in the message sets, listed in internal priority order: the highest to the lowest priority from top to the bottom of the list.

Set 1		Set 2		Set 3		Set 4		Set 5		Set 6		Set 7	
ID (decimal)	Message Period (ms)	ID (decimal)	Message Period (ms)	ID (decimal)	Message Period (ms)	ID (decimal)	Message Period (ms)	ID (decimal)	Message Period (ms)	ID (decimal)	Message Period (ms)	ID (decimal)	Message Period (ms)
5	6.25	5	6.25	5	6.25	5	12.5	5	12.5	5	12.5	5	12.5
1	6.25	6	6.25	6	6.25	6	12.5	6	12.5	6	12.5	6	12.5
2	6.25	7	6.25	7	6.25	1	12.5	1	12.5	1	12.5	1	12.5
3	6.25	8	6.25	8	6.25	2	12.5	2	12.5	2	12.5	2	12.5
4	6.25	10	6.25	10	6.25	3	12.5	3	12.5	3	12.5	3	12.5
6	12.5	1	6.25	1	6.25	4	12.5	4	12.5	4	12.5	4	12.5
7	12.5	2	6.25	2	6.25	7	25	7	25	7	25	7	25
8	25	3	6.25	3	6.25	8	25	8	25	8	25	8	25
10	50	4	6.25	4	6.25	10	25	10	25	10	25	10	25
9	25	9	6.25	9	6.25	17	50	17	50	17	50	17	75
11	50	11	6.25	11	6.25	18	100	18	50	18	50	18	75
12	100	12	12.5	12	6.25	20	100	20	100	20	50	20	75
13	100	13	12.5	13	6.25	21	100	21	100	21	50	21	75
		14	12.5	14	6.25	9	25	22	100	22	50	22	75
		17	25	17	12.5	11	25	23	100	23	750	23	100
		18	50	15	12.5	12	50	9	25	24	750	24	100
		20	50	16	12.5	13	50	11	25	25	750	25	100
		21	100	18	25	14	50	12	25	26	750	26	100
		22	100	20	25	15	50	13	50	28	750	28	100
		23	100	21	50	16	50	14	50	9	25	9	25
		15	25	22	50	19	100	15	50	11	25	11	25
		16	25	23	50			16	50	12	25	12	25
		19	50	24	100			19	100	13	25	13	25
				25	100					14	25	14	25
				26	100					15	25	15	25
				19	25					16	25	16	25
										19	50	19	75
										27	750	27	100

Figure 4.3: The Message IDs and Nominal Transmit Rates of Transmit Messages in Each Message Set, Listed in the Order of Internal Transmission Priority.

With no prior knowledge of the test bench system, these sets of transmit messages are designed based on the choices of the calibration values (total number of slots in the FIFO queues, number of reserved slots, *etc.*), and they differ in number of messages of each TX task in order to create different scenarios that can show the behaviors of the DFC stack under different levels of message load. The unique characteristics of each message set are shown in Figure 4.4, with the corresponding choices of the number of messages assigned to each of the TX tasks.

	Queue 0 maximum capacity: 11 messages	Queue 1 maximum capacity: 21 messages		
Message Set	Number of 6.25ms TX task messages	Number of 12.5ms TX task messages	Number of 25ms TX task messages	Unique characteristic
Set 1	5	2	6	Low message load in both queues, less than maximum capacity
Set 2	11	3	9	Number of Queue 0 messages at maximum capacity
Set 3	14	3	9	Number of Queue 0 messages over maximum capacity
Set 4	0	6	15	Number of Queue 1 messages at maximum capacity
Set 5	0	6	17	Number of Queue 1 messages slightly over maximum capacity
Set 6	0	6	22	Number of Queue 1 messages significantly over maximum capacity
Set 7	0	6	22	Same as Set 6 but different transmit rates for Queue 1 messages

Figure 4.4: Unique Characteristics of the Transmit Message Sets.

According to the configuration of the experiments, Queue 0 has a total number of 10 slots, and consequently the maximum capacity at any given time is 11 messages, including the TX buffer of Queue 0, which means that Queue 1 should be able to effectively transmit 11 messages after a series of consecutive enqueues in a TX task execution (first message would be loaded into the TX buffer directly since the queue is empty and the buffer is available). Similarly, Queue 1 has a maximum capacity of 21. The 6.25ms TX task is assigned with transmit messages in only message Sets 1 through 3, and the focus of these message sets is to analyze the behaviors of Queue 0 at different message loads. These message sets specifies different numbers of messages that are to be transmitted through Queue 0, more specifically, the maximum possible number of transmit-pending messages for Queue 0 during an instance of the

6.25ms TX task that is below, equal to and over the maximum capacity of Queue 0, respectively, in order to reflect the order of message enqueues and exactly which messages are enqueued in Queue 0. Besides the insights of Queue 0 behaviors, message Sets 1 through 3 also show behavior of Queue 1 when having a maximum number of transmit-pending messages that is below the maximum capacity. For messages Set 4 through 7, there are no messages assigned to the 6.25ms TX tasks and the focus is on the behaviors of Queue 1 when having a total number of messages that is equal to, slightly above and significantly above the maximum capacity.Note that message Set 7 has the same message IDs and internal transmission priories as Set 6, however different transmit rates are assigned to those message IDs, aiming to allow different subsets of the transmit messages to enter the reserved slots. Message Sets 4 through 7 are set out to create experiment scenarios that can reflect (i) the order of message enqueues, (ii) which messages are enqueued in each instance of their TX tasks, (iii) how messages are handled if they are not eligible for transmission after they become pending for transmission and (iv) the usage of reserved slots in Queue 1.

By having a Lauterbach hardware debugger connected to the hardware test bench, it is possible to record the contents of several memory locations of interest at run time, obtaining the raw trace files for each experiment scenario. In order to translate the recorded raw traces a into a more readable form, a parser program has been developed to parse the raw trace files, translate each raw trace into a trace of a series events of interest and then save the translated traces into comma-separated-values (CSV) files. A section of a post-parser trace is shown in Figure 4.5. In the first column, there is the unique number assigned to each event by the Lauterbach debugger. The second column shows the time stamps, in microseconds, of the captured events in absolute

time since the start of the test bench experiment. The third column ("Event Type") is a descriptive name of the event telling the type of the event (enqueue, dequeue or transmission-complete interrupt) and the transmit queue associated with that event. Message IDs of messages involved in events in Queue 0 are listed in Columns 4 and 5. A message ID in Column 4 indicates that an instance of a message with that ID has just entered Queue 0, and a message ID shown in Column 5 indicates that an instance of a message with that ID has just been loaded into the TX Buffer 0 for transmission. Similarly, Column 6 and 7 show the enqueued and dequeued message IDs of Queue 1. When a message bypasses its target queue and gets loaded into the corresponding TX buffer of the target queue (if the target queue is empty and its corresponding TX buffer is available when a message is assessed for queuing), it is shown as a dequeue event, and the message ID of that message will be displayed in either Column 5 or 7 depending on the target queue. Such a queue-bypass event differs from a regular dequeue event in the sense that there is not a previous enqueue event with the same dequeue-event message ID. For a transmission-complete ISR event, the ID of the message, which has just been transmitted on the bus and invokes the ISR, is shown in Column 8, the last column.

Event ID	Time [us]	Event Type	Q0 Enqueue	Q0 TX Buffer	Q1 Enqueue	Q1 TX Buffer	ISR
-1097865	116985	Q0_TxBuff		5			
-1097854	116994	Q0_Slot5	1				
-1097841	117001	Q0_Slot6	2				
-1097829	117009	Q0_Slot7	3				
-1097817	117016	Q0_Slot8	4				
-1097805	117217	ISR					5
-1097799	117219	Q0_TxBuff		1			
-1097788	117464	ISR					1
-1097782	117465	Q0_TxBuff		2			
-1097779	117564	ISR					2
-1097773	117566	Q0_TxBuff		3			
-1097762	117806	ISR					3
-1097755	117808	Q0_TxBuff		4			
-1097744	118048	ISR					4
-1097740	120675	Q1_TxBuff				6	
-1097731	120685	Q1_Slot9			7		
-1097719	120888	ISR					6
-1097713	120891	Q1_TxBuff				7	
-1097702	121138	ISR					7
-1097698	121244	Q1_TxBuff				8	
-1097693	121253	Q1_Slot10			9		
-1097681	121378	ISR					8
-1097675	121380	Q1_TxBuff				9	
-1097668	121550	ISR					9
-1097664	122861	Q0_TxBuff		5			
-1097653	122869	Q0_Slot9	1				
-1097639	122877	Q0_Slot10	2				
-1097627	122885	Q0_Slot1	3				

Figure 4.5: A Section of the Parsed Trace Data of a Test Bench Experiment.

4.2 Observations about Test Bench Behavior and Bench Data

Compared to the specification documents of the DFC stack, unexpected timing behaviors have arisen in the experiment data obtained from the hardware test bench. These behaviors do not conflict with the specification documents of the DFC stack however are not described in the documents. This section will identify unexpected data patterns observed in test bench traces about the test bench (RTOS and hardware setup) and DFC stack but are not described in the specification documents. After careful analysis of the experiment data, it is concluded that the unexpected behaviors arise due to the characteristics of the host environment, more specifically the performance limitation of the test bench hardware and the internal implementation of the RTOS. The patterns of such behaviors provide insights to how the host environment effects the timing of different types of events in the DFC stack.

For generating realistic simulation results, it is crucial to identify the causes of these patterns and then implement corresponding configurable features in the simulation model so that the timing effects introduced by the host environment can be reflected in the simulation results. Suitable models of the timing effects need to be created based on the test bench trace data, and will be used as input parameters to the configurable features to recreate the characteristics of the host environment. As a beneficial side effect, the DFC stack simulation model will be able to simulate the DFC stack running on other host environment if it is possible to determine the input parameters of the target host environment for these configurable features.

This section discusses the patterns and the impact of the timing effects introduced

by the host environment, and explains the importance of incorporating such system details in the simulations, while more details on the how they are modeled will be provided in Section 6.

4.2.1 Message Enqueue Time

Based on the time stamps of consecutive enqueue events in the test bench traces, there is a time gap between each enqueue event of a transmit message and its previous event. The duration of such time gaps is the "regular enqueue time" of a message, which is the total amount of time it takes for the queuing assessment (by the CAN Handler) of the message and loading the message into its target queue (by the CAN Device Driver). A duration of regular enqueue time is mandatory for loading any transmit-pending message into its target queue.

As mentioned earlier, one of the cores of the dual-core MCU is assigned to handle the enqueuing of transmit messages while the other is for the dequeuing of messages. There is a semaphore that protects the critical sections of the enqueue and dequeue events. However, once exiting the critical section of the enqueue process of a message (near the beginning of the duration of the enqueue process), it is possible that a transmission-complete ISR is invoked and stalls the enqueuing process. In such scenarios, where the transmission-complete ISR is invoked during the enqueue process of a message, the bench data shows that the total duration of the message enqueue (the total duration that the enqueue core spends of both before and after the interruption of the transmission-complete ISR) is extended compared to a regular enqueue process. The total duration of this interrupted enqueue process is called the "irregular enqueue time", as shown in Figure 4.6.

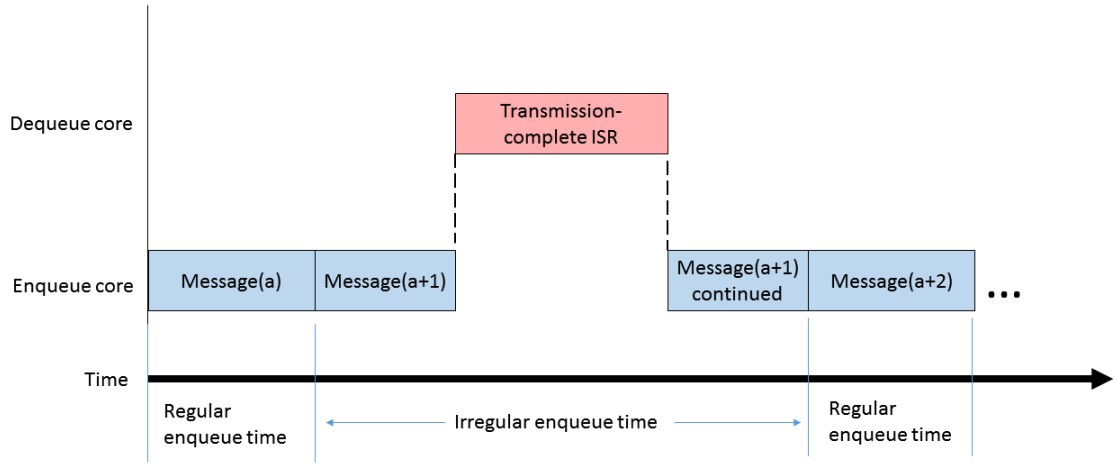

Figure 4.6: Regular and Irregular Enqueue Times

Based on the bench data, it appears that the duration of the irregular enqueue time is generally longer than the combined duration of the execution time of the transmission-complete ISR and the regular enqueue time for a single message, which is mostly likely due to the overhead work (context switch in processor, *etc.*) that the RTOS has to perform when switching back and forth between the enqueue and dequeue tasks.

Figure 4.7 shows a comparison of message enqueue sequences between applying minimal regular enqueue times and relatively longer enqueue times in a hypothetical situation where there are a total of 7 transmit-pending messages at time 0 and only 6 empty slots left in the queue.

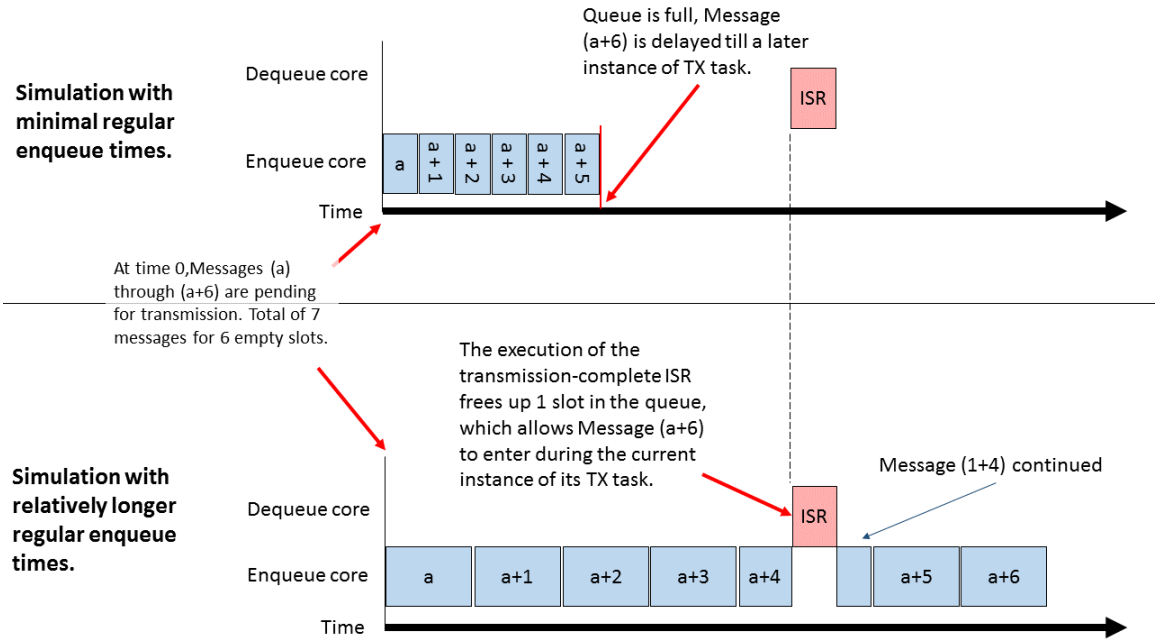

Figure 4.7: Impact of Regular Enqueue Times in Message Output Sequence

A blue box in the enqueue core region indicates the enqueue process of a message and the width of the box is the duration of the enqueue time. A red box in the dequeue core region indicates the execution of the transmission-complete ISR, during which a message is removed from the queue and loaded into the TX buffer of that queue.

When having minimal regular enqueue times (top part of the figure), the queue will be filled up by Messages (a) through (a + 5) and there is no space left in the queue for Message (a + 6). A transmission-complete ISR is invoked after the enqueue of Message (a + 5), but since the TX task has already finished executing, Message (a + 6) will be delayed and assessed again for queuing during the next instance of its TX task. However, if there are longer than regular enqueue times for messages (the bottom part of the figure), because the transmission-complete ISR is invoked, for

example during the enqueuing of Message (a + 4), it is executed before the queuing assessment of Message (a + 6), as it takes longer to load messages into the queue. Consequently, a queue slot is freed up by the execution of the transmission-complete ISR so there are enough slots in the queue for all of the transmit-pending messages in the current instance of the TX task to be enqueued, including Message (A + 6). As a result, when having relatively longer durations for the regular enqueue times, the messages that are transmitted on the bus during the current instance of the TX task are different than in the case where there were only minimal regular enqueue times.

To demonstrate the impact that the extended duration of an irregular enqueue has on the message output during the current instance of the TX task, Figure 4.8 shows a hypothetical situation where there are 7 transmit-pending messages at time 0 and only 5 empty slots left in the queue.

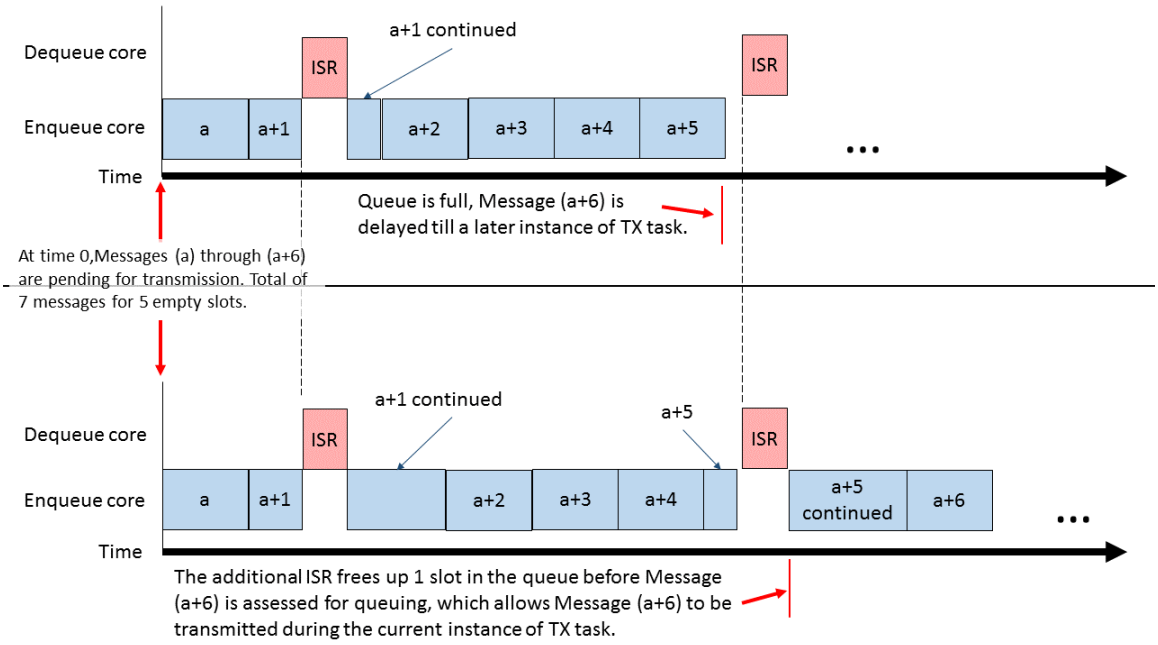

Figure 4.8: Impact of Irregular Enqueue Times on Message Output Sequence

A transmission-complete ISR is invoked during the enqueue of Message (a + 1), which frees up a slot in the queue and makes it 6 available slots for 7 transmit-pending messages. If the total duration of the enqueue time of a message is not affected by the transmission-complete ISR's (no extended duration for an irregular enqueue, as shown in the top part of the figure), by the end of the current instance of the TX task, Messages (a) through (a + 5) successfully enter the 6 available slots in the queue but Message (a + 6) is delayed until a later instance of the TX task due to a full queue. However, if irregular enqueue time is introduced by the transmission-complete ISR during the enqueuing of Message (a + 1), the enqueue of Message (a + 5) is delayed, which allows the execution of an additional transmission-complete ISR to take place before Message (a + 6) is assessed for queuing. Execution of the additional transmission-complete ISR frees up a slot in the queue for Message (a + 6) so that Message (a+6) can enter the queue during the current instance of the TX task.

In summary, the durations of the regular and irregular enqueues can affect the number of messages that can be transmitted during the current instance of the TX tasks. It is essential to model these durations in order to produces realistic simulation results that can reflect the behaviors of the actual system.

4.2.2 Inter-TX-Task Delay

As the DFC stack is a part of an RTOS which runs many tasks, and it is possible that jitters and delays are introduced into the execution of DFC stack related tasks due to the arrivals of higher-priority tasks in the RTOS. The test bench traces revealed time gaps in-between the execution of the 3 TX tasks of the DFC stack, when more

than 1 of the TX tasks arrive at the same time in all of the experiment scenarios. The cause is that there are tasks from other parts of the RTOS constantly being scheduled in-between the 3 TX tasks of the DFC stack. These inter-TX-task delays, during which external tasks are being executed, are called "t_i", time of inactivity, as shown in Figure 4.9, where $t_{i,1}$ denotes time of inactivity between the 6.25ms and 12.5ms TX tasks and $t_{i,2}$ denotes time of inactivity between the 12.5ms and 25ms TX tasks.

Due to the fact that the periods of the 12.5ms and 25ms TX tasks are multiples of 2 and 4 of the period of the 6.25ms TX task, respectively, whenever the 25ms TX task is released, the other two TX tasks are also released at the same time. As a result, there are only $t_{i,1}$ and $t_{i,2}$, but there is no such thing as a t_i between the 6.25ms and 25ms TX tasks.

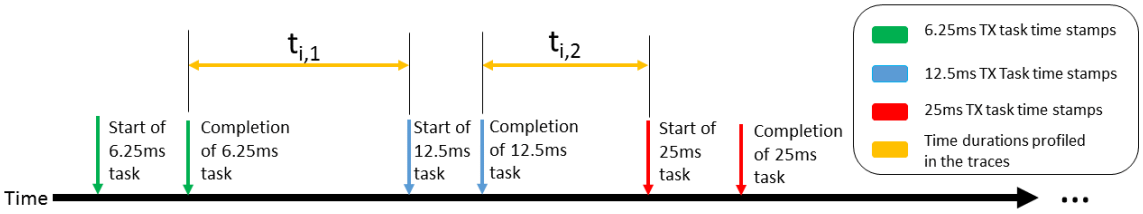

Figure 4.9: Time of Inactivity in-between TX Tasks

The durations of $t_{i,1}$'s and $t_{i,2}$'s have been profiled from test bench traces of all 14 experiment scenarios. Each occurrence of a $t_{i,1}$ is computed by taking the backward time difference between the time stamp of the first dequeue event of the 12.5ms TX task (it always showed up as a dequeue event since the first message of the 12.5ms always bypassed the FIFO queue due to Queue 1 being empty and an available hardware TX buffer) and the time stamp of the last enqueue event of the 6.25m TX

task. The $t_{i,2}$'s are computed in the same fashion between the 12.5ms and 25ms TX tasks (however, the first event of the 25ms TX task may be either an enqueue or dequeue event depending if there are still 12.5ms TX task messages left in their target FIFO queue and TX buffer). The computed $t_{i,1}$'s and $t_{i,2}$'s from each individual trace are shown in Figure 4.10 and Figure 4.11, respectively. Note that for $t_{i,1}$, results are available for only message Sets 1 through 3, because there are no transmit messages in the 6.25ms TX task for message Sets 4 through 7.

(unit: ms)		Message Set 1	Message Set 2	Message Set 3
Configuration: 18 non-reserved slots in Queue 1, threshold for entering: 75ms	Number of occurrences	74	74	74
	Average (arithmetic mean)	3.660675676	3.652040541	3.653851351
	Maximum	3.744	3.759	3.755
	Minimum	3.608	3.587	3.589
	Standard deviation	0.039585215	0.04737592	0.043310818
(unit: ms)		Message Set 1	Message Set 2	Message Set 3
Configuration: 19 non-reserved slots in Queue 1, threshold for entering: 50ms	Number of occurrences	74	74	74
	Average (arithmetic mean)	3.660324324	3.651094595	3.653378378
	Maximum	3.758	3.738	3.753
	Minimum	3.602	3.59	3.588
	Standard deviation	0.04063675	0.044981631	0.045127734

Figure 4.10: Profiled $t_{i,1}$ Between 6.25ms TX Task and 12.5ms TX Task

(unit: ms)		Message Set 1	Message Set 2	Message Set 3	Message Set 4	Message Set 5	Message Set 6	Message Set 7
Configuration: 18 non-reserved slots in Queue 1, threshold for entering: 75ms	Number of occurrences	37	37	37	37	37	37	37
	Average (arithmetic mean)	0.559918919	0.54527027	0.54427027	0.55372973	0.553594595	0.553648649	0.55345946
	Maximum	0.563	0.548	0.547	0.559	0.558	0.559	0.56
	Minimum	0.557	0.542	0.54	0.55	0.551	0.55	0.548
	Standard deviation	0.001639417	0.001484151	0.001693921	0.00177402	0.001571614	0.001888801	0.002022026
(unit: ms)		Message Set 1	Message Set 2	Message Set 3	Message Set 4	Message Set 5	Message Set 6	Message Set 7
Configuration: 19 non-reserved slots in Queue 1, threshold for entering: 50ms	Number of occurrences	37	37	37	37	37	37	37
	Average (arithmetic mean)	0.560324324	0.545216216	0.544432432	0.553756757	0.553243243	0.55345946	0.553540541
	Maximum	0.564	0.548	0.547	0.56	0.56	0.561	0.56
	Minimum	0.557	0.54	0.54	0.551	0.549	0.548	0.55
	Standard deviation	0.001667567	0.001781621	0.001833948	0.001770207	0.002350005	0.002049317	0.001741561

Figure 4.11: Profiled $t_{i,2}$ Between 12.5ms TX Task and 25ms TX Task

The durations of the $t_{i,1}$'s appear to be very close to one another across all experiment scenarios, and so are the durations of the $t_{i,2}$'s. Therefore, the statistical model of each t_i is created from the durations of all its occurrences in the 14 experiment scenarios. More details on how the t_i's are modeled will be provided later in Section 6.2. For demonstration purposes, Figure 4.12 shows the computed parameters (arithmetic mean, standard deviation, *etc.*) for the Gaussian distribution models of $t_{i,1}$ and $t_{i,2}$.

(unit: ms)	$t_{i,1}$	$t_{i,2}$
Total number of occurrences	444	518
Average (arithmetic mean)	3.655227478	0.551990348
Maximum	3.759	0.564
Minimum	3.587	0.54
Standard deviation	0.043509113	0.005382641

Figure 4.12: Parameters for Gaussian Distribution Statistical Models of $t_{i,1}$ and $t_{i,2}$.

Due to these inter-TX-task delays, the enqueue of the messages from 12.5ms and 25ms TX tasks are delayed to a later point in time after they become pending for transmission, which consequently affects the arrival times of these messages on the bus. Therefore, it is essential to model the t_i delays in the DFC stack simulation model in order to achieve a high level of fidelity. Figure 4.13 shows the effect of the inter-TX-task delays (more specifically, $t_{i,2}$'s in this example) in a hypothetical but realistic situation, in which:

- There are a total of 25 transmit-pending messages from the 12.5ms and 25ms TX tasks that have just been released.
- All the transmit-pending messages are to enter Queue 1 for transmission.
- The total number of available slots in Queue 1 is 20, and these slots are all non-reserved slots.

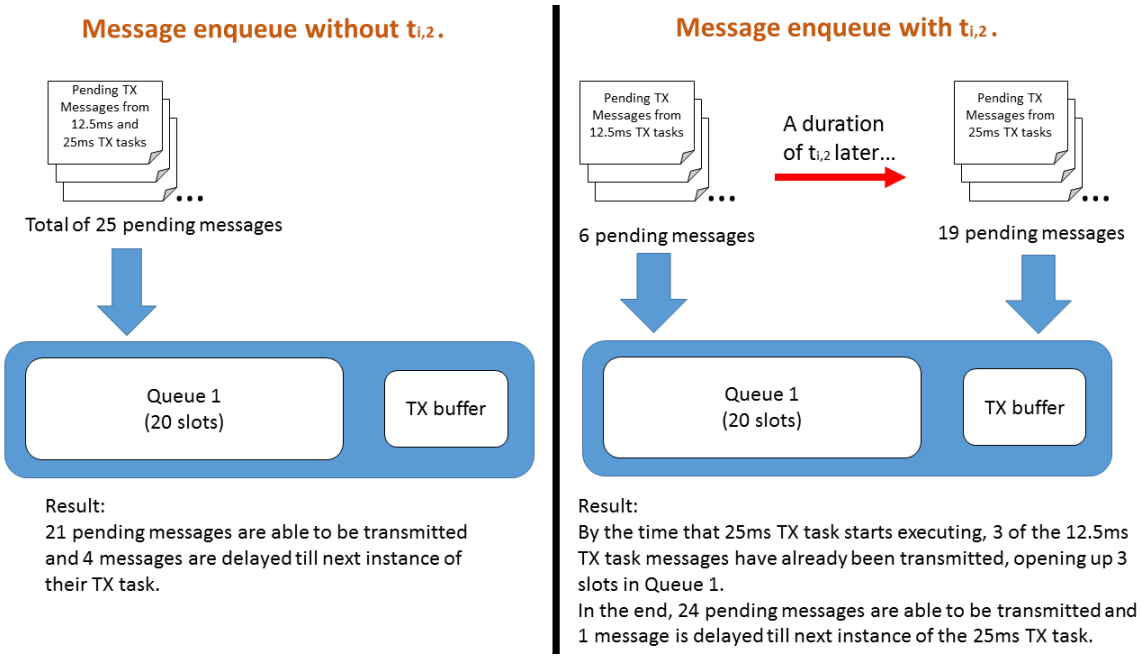

Figure 4.13: Effect of Non-zero Inter-TX-Task Delay($t_{i,2}$) on Message Enqueuing

Without $t_{i,2}$, the 25ms TX task messages will be assessed for queuing immediately after the 12.5ms TX task, which means all 25 messages are to be assessed very closely in time. However, Queue 1 and its TX buffer can only take in up to 21 messages (including the case where the first message bypasses the queue and enters a TX buffer directly due to an empty queue and available TX buffer). This would only allow 21 out of 25 messages to be transmitted at the current instance of their TX tasks, and the 4 messages that are not transmitted will not be assessed again until the next instance of their TX tasks, which would delay the arrivals on bus of these 4 messages for at least the duration of one period of their TX task.

On the other hand, if the RTOS delays the 25ms TX task for a duration of $t_{i,2}$ after the 12.5ms TX task, as happens in the test bench experiments, it is very likely that a number of the enqueued 12.5ms TX task messages have been transmitted during

the delay. If 3 of the 12.5ms TX task messages have been transmitted on the bus, 3 slots that were occupied by those 3 messages will become available in the queue for the 25ms TX task messages, which would allow 3 more 25ms TX task messages to be transmitted during the current instance of 25ms TX task. As a result, 24 out of 25 messages will be transmitted and 1 message is delayed till the next instance of the 25ms TX task, instead of 21 messages transmitted and 4 messages delayed messages when there is no $t_{i,2}$. This would consequently create a different message sequence on the CAN bus as the possibly delayed messages will appear on the bus after different instances of the 25ms TX task.

4.2.3 Measurement Jitter in Message Bus Times

From the experiment trace data, it is possible to proximate the bus time (duration of a successful transmission of a complete CAN frame on the bus) of each instance of the transmit messages. Since the DFC stack ECU is the only transmitting node in the CAN network, whenever a message starts transmission on the bus after being loaded into a TX buffer, that message will automatically win arbitration with no collision on the bus, and finishes transmission on its first attempt. Therefore, the bus time of a message in the test bench experiments ("recorded bus time") can be calculated by taking the backward time difference between the message's transmission-complete ISR event and its dequeue event (message loaded into TX buffer).

Theoretical bus times of the transmit messages are calculated and are compared with the recorded bus times in the test bench traces. Since arbitrary data is being sent in the data frames during the test bench experiments, it is possible that the bus times of the messages are under the effect of bit-stuffing. The bus time of a CAN data

frame is calculated by dividing the number of bits by the baud-rate of the bus. For a CAN data frame with a standard 11-bit identifier (all message in the experiments use standard identifiers), the total length is the sum of 47 bits (required structure of the data frame format) and the bits in the data field. This gives the formula for bus time without any bit stuffing as:

$$t_{bus} = \frac{47 + 8 * DLC}{baudrate}$$

The DLC field of a message DLC of indicates the number of bytes in the data field and there are 8 bits in each byte of data. Out of the required 47 bits, 33 bits are prone to bit stuffing. Therefore the total number of bit that are prone to bit-stuffing is $(33 + (8 * DLC))$. For worst-case bit stuffing (wcbs), as introduced in Section 2.1, 1 bit is stuffed for every 4 bits. Therefore the total number of bits under worst-case bit stuffing is:

$$47 + (8 * DLC) + \lfloor 0.25 * (33 + 8 * DLC) \rfloor$$

The bus time of a data frame under worst-case bit stuffing is then:

$$t_{bus,wcbs} = \frac{47 + (8 * DLC) + \lfloor 0.25 * (33 + 8 * DLC) \rfloor}{baudrate}$$

During the experiments on the hardware test bench, the influence of the characteristics of the physical CAN bus and electromagnetic interference is assumed to be insignificant and is ignored, it is safe to assume the actual bus times of messages transmitted by the DFC stack ECU are very close to their theoretical bus times.

The recorded bus times of the messages in each experiment scenario are compared to the theoretical bus times (calculated values of the theoretical bus times). A section

of the bus time comparison chart is shown in Figure 4.14. The top row with yellow background shows the message information and theoretical bus times with and without worst-case bit stuffing. The blue-background and green-background data show the recorded bus times of the same message set under two different configuration of the DFC stack. As each message appears many times during an experiment run, only the average (arithmetic mean), maximum and minimum of the recorded bus times are shown in the figure. The bounds of the measurement jitter between the recorded bus times and theoretical bus times are shown in the bottom two rows.

Message Info. & Theoretical Bus Time	Bus Time Unit: us	Message ID (decimal), DLC, Bus time no bit-stuffing, bus time worst-cast bit-stuffing	1, 8, 222, 270	2, 0, 94, 110	3, 8, 222, 270	4, 8, 222, 270	5, 8, 222, 270	6, 6, 190, 230	7, 8, 222, 270	8, 2, 126, 150	9, 4, 158, 190	10, 0, 94, 110
Test Bench Experiment Scenario 5	18 non-reserved slots in Queue 1, period threshold for entering: 75ms	Average	244.622	98.6486	240.703	240.608	232.297	212.824	264.297	160.189	177.946	119.541
		Maximum	247	107	243	244	236	216	346	211	218	145
		Minimum	242	98	240	240	230	210	246	133	170	98
	19 non-reserved slots in Queue 1, period threshold for entering: 50ms	Average	244.622	98.6081	240.649	240.581	232.27	212.932	264.216	159.865	178.135	121.189
		Maximum	250	105	243	243	237	215	344	214	220	144
		Minimum	242	98	240	240	230	211	246	132	170	97
		Jitter upper bound	28	13	21	22	15	26	124	88	62	51
		Jitter lower bound	20	4	18	18	8	20	24	6	12	3

Figure 4.14: A Section of the Bus Time Comparison Chart.

It appears that, even though the durations of recorded bus times are generally shorter than the theoretical bus times under worst-case bit stuffing, they are still considerably longer than the theoretical bus times without any bit stuffing, for all transmit messages in all 14 experiment scenarios. It is concluded that there is jitter between the actual bus times of the messages and the recorded bus times that are computed from the test bench traces. The jitter in the recorded bus time is called the "measurement jitter", and it is most likely caused by software, more specifically, the

RTOS and/or DFC stack itself. Figure 4.15 shows the relationship between the actual bus time, the measurement jitter and the recorded bus time of a transmit message in the DFC stack.

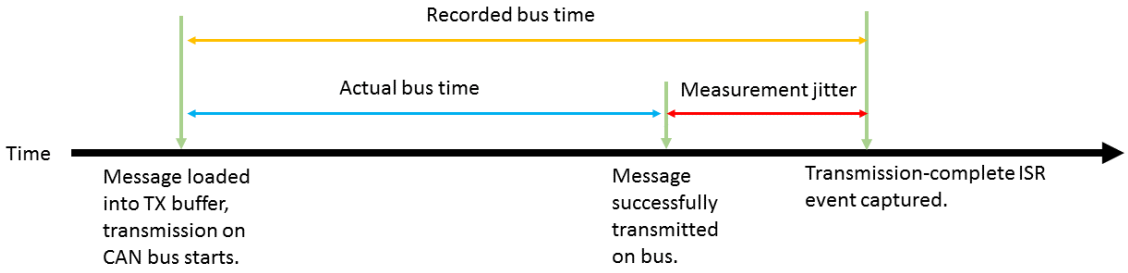

Figure 4.15: The Actual Bus Time and the Measurement Jitter in the Recorded Bus Time of a Message.

Although it is not clear what exactly causes the jitter, Figure 4.16 shows several possible causes, in a zoomed-in illustration of what might causing the duration labeled "measurement jitter" in Figure 4.15. After a message finishes transmission on the bus and the CAN Device Driver invokes the transmission-complete ISR, the RTOS may be busy with some other higher-priority task and the ISR has to wait until the RTOS responds. Once RTOS decides to service the transmission-complete ISR, it will prepare the MCU through a context-switch. After the ISR starts execution, there may be other work that the ISR needs to handle before it writes the message ID of the transmitted message to memory, where the debugger monitors and captures the ISR event when a new message ID is written. And during the execution of the transmission-complete ISR, it may be interrupted by the RTOS to execute a higher-priority ISR from other parts of the RTOS.

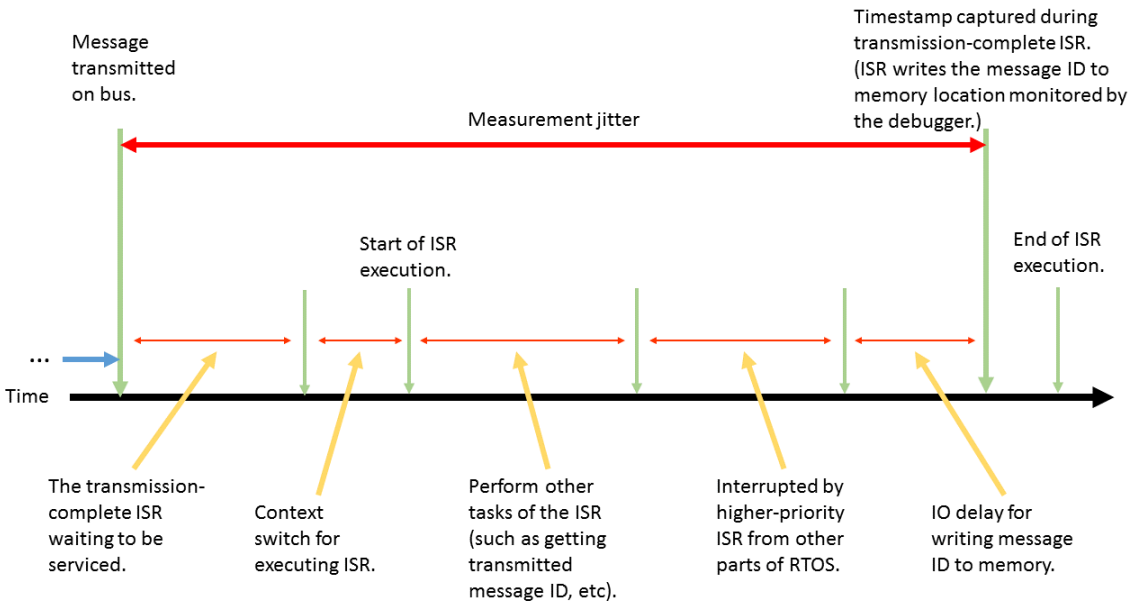

Figure 4.16: The Possible Components of the Measurement Jitter

Figure 4.17 shows a comparison of message throughput between the test bench experiment, where there is measurement jitter, and a hypothetical system, where the durations of the measurement jitter are only minimal. Each tall green arrow in the figure indicates the start of transmission of a message (message ID labeled below the time axis).The gap between the 2 tall green arrows indicates the duration of the message transmission that is measurable from the experiment traces, or in other words, the recorded bus time of that message. The actual durations that it takes for the TX buffer to transmit any message in this example, or the actual bus time of messages, are assumed constant.

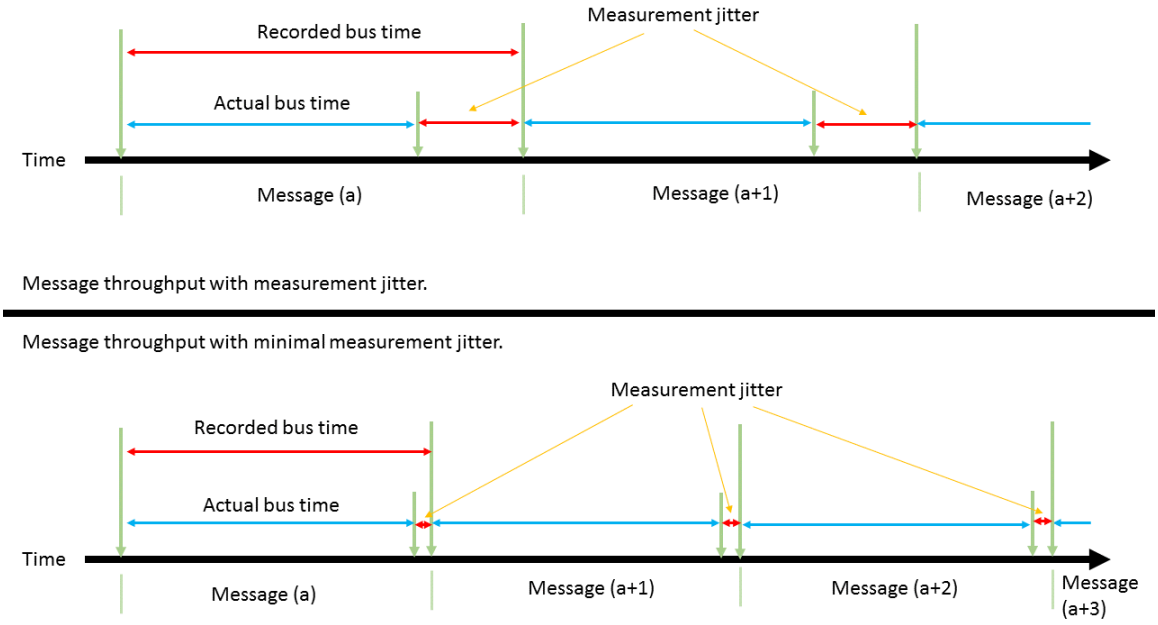

Figure 4.17: Measurement Jitter's Impact on Message Throughput

Compared to the message throughput at the top of the figure, minimal measurement jitter allows the messages in the queue to leave the queue sooner, which frees up some of the occupied queue slots for the remaining transmit-pending messages before they are assessed for queuing. Consequently, this will potentially allow more messages to enter the queue during the current instance of their TX tasks. In other words, when a large number of messages are pending for transmission, having smaller durations of measurement jitter is likely to allow more transmission-pending messages to enter the queue, and consequently produce a different message sequence on the bus compared to having longer measurement jitter durations.

The measurement jitter can have great impact on the output message sequence on the CAN bus, and it is necessary that the DFC stack incorporates the measurement jitter in the implementation to generate more realistic simulation results.

Chapter 5

The Simulation Model

This section will introduce the DFC stack simulation model in detail, covering aspects such as the components of the model, design decisions and MATLAB SimEvents features that are used to develop the model.

In order to achieve a high level of configurability, the simulation model should be able to accept input parameters that provide necessary information for the target simulation setup, more specifically, information containing the configuration of the DFC and a description of the simulated host environment. Consequently, the DFC stack simulation model is implemented as two parts, the simulation script and the DFC stack SimEvents model. The simulation script provides access to the configurable input parameters for the simulation and is the user interface for using the SimEvents model. To start a simulation, the user can simply configure the input parameters in the simulation script and then run the simulation script, which loads the input parameters into the MATLAB environment, runs the simulation on the SimEvents model and processes the simulation results once the simulation completes.

5.1 The Simulation Script

The simulation script is a MATLAB script and it consists of two sections.

The first section contains the configurable input parameters of the DFC stack SimEvents model. These input parameters can be divided further into three categories, more specifically, ones that are relate to the configuration of the DFC stack, ones that provide information about the host environment to be simulated and ones that specify how the simulations are to be conducted. The input parameters are saved as MATLAB workspace variables during simulations so that they can be accessed by other scripts and SimEvents blocks in the simulation model.

The second section of the simulation script contains MATLAB instructions that setup and start the simulation, as well as calls to other scripts that process the simulation results.

5.1.1 Configurable Input Parameters

The first category of the input parameters contains calibrations (settings) for the DFC stack and information about the transmit messages.

Each combination of values for the calibrations forms a unique configuration of the DFC stack. Although the simulation model is capable of simulations with any values for the calibrations, in order to validate the simulation model against the test bench experiments (Section 6), there are 2 pre-defined sets of calibration values in the simulation script that correspond to the 2 configurations that are used during the test bench experiments. The first set of calibrations specifies 18 non-reserved slots in Queue 1 and 75 milliseconds as the message transmit rate threshold for entering the reserved slots (Section 3.2.1), while the second set specifies 19 non-reserved slots and

50 milliseconds as the transmit rate threshold. Aside from these 2 calibrations, other calibrations are identical in the pre-defined sets of calibrations, more specifically, zero reserved slots in Queue 0 and 6.25 milliseconds as transmit-rate threshold for entering Queue 0 (i.e., only 6.25ms TX task messages will be able to enter Queue 0, all 12.5ms and 25ms TX task messages will be routed to Queue 1).

The information about the transmit messages is required to run a simulation, and it is defined in terms of a message set in the simulation script. For each message set, the message ID, DLC, nominal transmit rate and internal transmit priority have to be provided separately for each message. For validation purposes of this thesis, 7 sets of transmit messages are pre-defined in this section of the simulation script. Although the specifics of these message sets are identical to that of the test bench experiments, the simulation model is capable of running simulation with any combinations of message IDs, DLCs, *etc.* for the transmit messages. As an example, Figure 5.1 shows the definition of Message Set 3 of the pre-defined transmit message sets. For convenience, the messages belonged to the 6.25ms, the 12.5ms and the 25ms TX tasks are referred to as the Bucket 1, Bucket 2 and Bucket 3 messages, respectively, in the SimEvents model, and they are further abbreviated as “B1”, “B2” and “B3” messages in the simulation script.

```
NUM_B1_MSG = 14;
B1_ID = [5 6 7 8 10 1 2 3 4 9 11 12 13 14]';
B1_PRD = repmat(6.25, 1, NUM_B1_MSG)';
B1_DLC = [8 6 8 2 0 8 0 8 8 4 0 0 0 0]';

NUM_B2_MSG = 3;
B2_ID = [17 15 16]';
B2_PRD = repmat(12.5, 1, NUM_B2_MSG)';
B2_DLC = [8 0 7]';

NUM_B3_MSG = 9;
B3_ID = [18 20 21 22 23 24 25 26 19]';
B3_PRD = [25 25 50 50 50 100 100 100 25]';
B3_DLC = [8 8 7 1 7 8 7 0 8]';
```

Figure 5.1: Definition of Message Set 3 in the Simulation Script.

The message IDs of each message bucket are listed in an array in the order of their internal transmit priorities: highest to the lowest from left to right (or, from the first to the last element in the array). The nominal transmit rates ("PRD", period) and the DLCs of the same message bucket are listed in the same order so that each value in these arrays is at the same position as its corresponding message ID in the message ID array.

The second category of the configurable input parameters contains the information for modeling the timing effects that are introduced by the host environment (discussed in Section 4.2). These input parameters will allow the simulation model to recreate the timing effects introduced by the host environment. The simulation model is capable of modeling the timing effects based on both deterministic and statistical models of their durations, and the available modeling options are provided in this section of the simulation script. The duration of each type of timing effects can be separately

set to either a constant value (deterministic model) or a random value generated based on a specified probability distribution (statistical model) through the use of pseudo-random number generation in MATLAB. The supported types of probability distributions include normal distribution, gamma distribution and kernel distribution. An addition modeling option is available besides using probability distributions, which is to randomly pick a value from the actual data of the durations, giving equal chance to every entry in the data. By having a variety of modeling options, the user can choose the one that best resembles each type of timing effect in order to increase the fidelity of the simulated host environment.

When using a probability distribution to generate the duration of a specific type of timing effect, the parameters of that probability distribution need to be entered into this section of the simulation script. For instance, the arithmetic mean and standard deviation are required for using the normal distribution, the scale and shape values are required for using the gamma distribution, *etc.*. The input parameters for the timing effects (introduced by the test bench) for these supported probability distributions have been computed from the test bench experiment traces and are provided in this section of the simulation script as pre-defined modeling options (more details on how they are computed are provided in Section 6.2).

The third category of the input parameters lists various options for automated simulations, including the duration of each simulation run, number of simulation runs for the current configuration, the types of simulation results that are to be processed and plotted after the simulations, and multiple sets of pre-defined values for previous categories of input parameters (DFC stack calibration values, transmit message set and parameters for the simulated host environment). The user can simply specify

the one of the available choices for each option here so that the simulation script can later perform simulations according to the specified options. For instance, the user can command the simulation model to plot the FIFO queue depths versus time graph or the inter-arrival times of the transmit messages on the bus after the simulations. For model validation purposes, the pre-defined values for the input parameters are identical to the test bench experiments (more details are in Section 6).

5.1.2 Simulation Instructions

The MATLAB instructions in this section of the simulation script automates the simulations and the processing of simulation results.

The instructions first read the specified indices for the input parameters and load the corresponding sets of values into the MATLAB workspace under variable names that are recognized by the SimEvents model. Then, the SimEvents model is started based on the specified duration of simulation and number of simulation runs. In between the simulation runs, the simulation instructions perform tasks such as shuffling and seeding the pseudo-random number generator, cleaning up and setting up temporary workspace variables, *etc.*.

Depending on the chosen options for simulation results in the input parameter section, the simulation instructions will selectively execute the supporting scripts at different times during the entire simulation procedure. For instance, the script, which computes and analyze the transmit rates of message based on the model generated trace, is called by the simulation instructions after each simulation run in order to record the transmit rate results from every single simulation run, while another script that formats and saves the transmit rate results is only called once after the entire

simulation procedure is completed.

5.1.3 Supporting Scripts

In order to present useful simulation results to the user, many lines of MATLAB instructions are used beside the 2 sections of the simulations script mentioned above. They perform tasks such as processing the simulation results, plotting results in figures and charts, saving results to files, *etc.* These MATLAB instructions have relatively more complex structures and are larger in volume than the rest of the simulation script, and they are low-level instructions that most likely do not interest the user. Therefore, in order to make the simulation script a straight-forward user interface, instead of having these instructions in the simulation script, they are organized into separate and smaller scripts with more readable natural-language names, and are called by the simulation script when necessary. The scripts are later referred to as the "supporting scripts", and are considered as a part of the simulation script.

5.2 The SimEvents Model

The DFC stack SimEvents model is contained in a SimEvents block that has three ports for the message entities to flow in and out. Transmitted messages entities leave the model through the TX port, and any received messages enter the model from the RX port. In order to allow integration of the DFC stack model with the Vector CAN stack library Whinton (2016), the DFC stack SimEvents model also has a Return port aside from TX and RX ports. The Return port will allow message entities, which are originally transmitted by the DFC stack model on the CAN network, to return to the

model to complete its transmission.

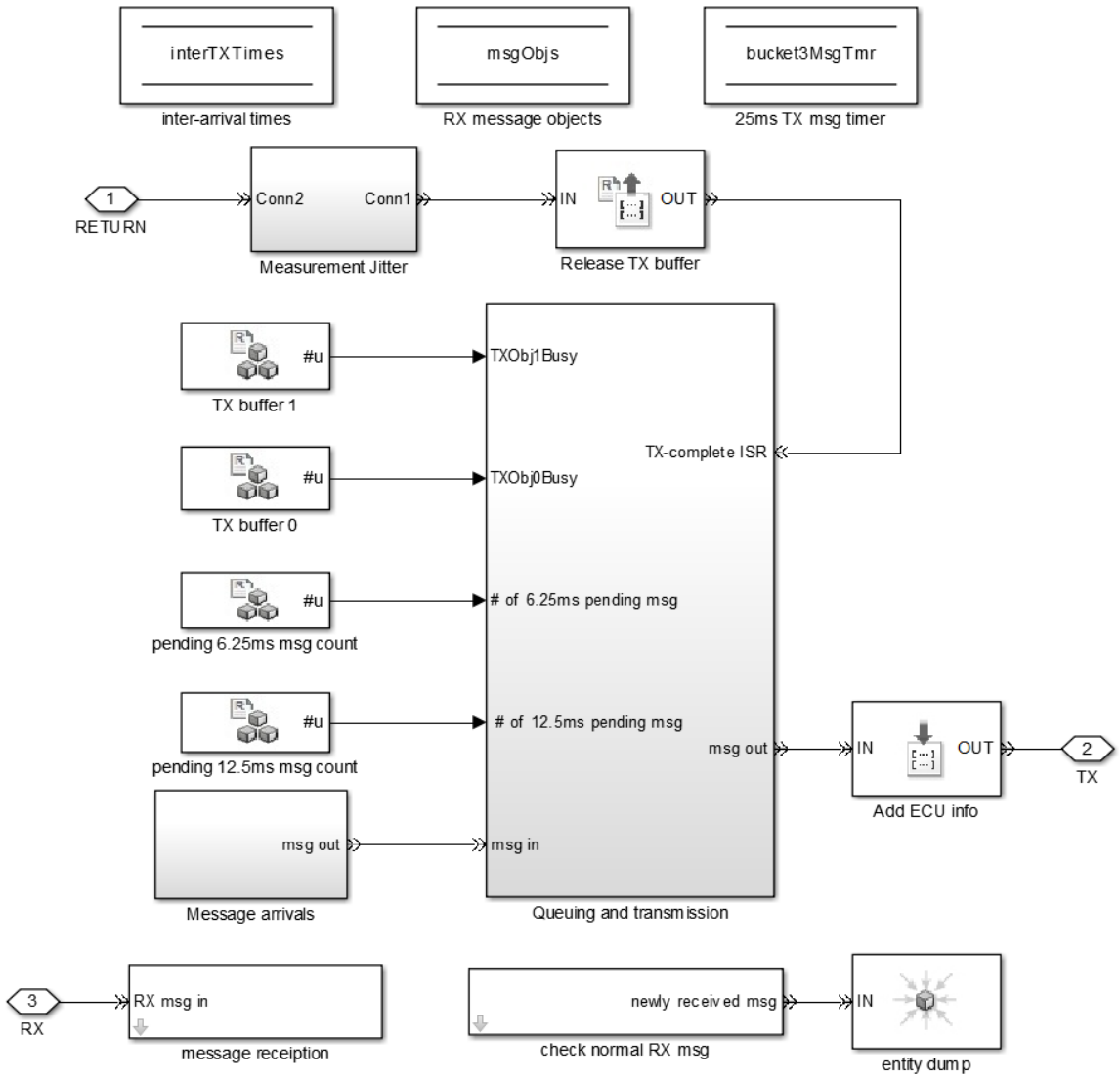

Figure 5.2: The Root Level Components of the SimEvents Model

Figure 5.2 shows the components in root level of the SimEvents model. The transmit messages propagates through two major components: new instances of the messages are generated in the Message Arrivals subsystem block, they are queued and/or loaded into TX buffers in the Queuing and Transmission subsystem block,

and then they leave the DFC SimEvents model through the TX port for transmission on the CAN network. Returned (or, successfully transmitted) messages re-enter the Queuing and Transmission block to complete their transmissions. There are also other SimEvents blocks in the root level of the model: the Global Data Store Memory blocks (labeled "inter-arrival times", etc) and the Resources Pool blocks (labeled "TX buffer 1", etc), and they will be introduced in detail in the following sections.

To illustrate the propagation of a message entity inside the DFC stack SimEvents model, Figure 5.3 shows the activity flow of a message instance, including the transitions and activities of that message from the creation of its message entity to the departure from the model.

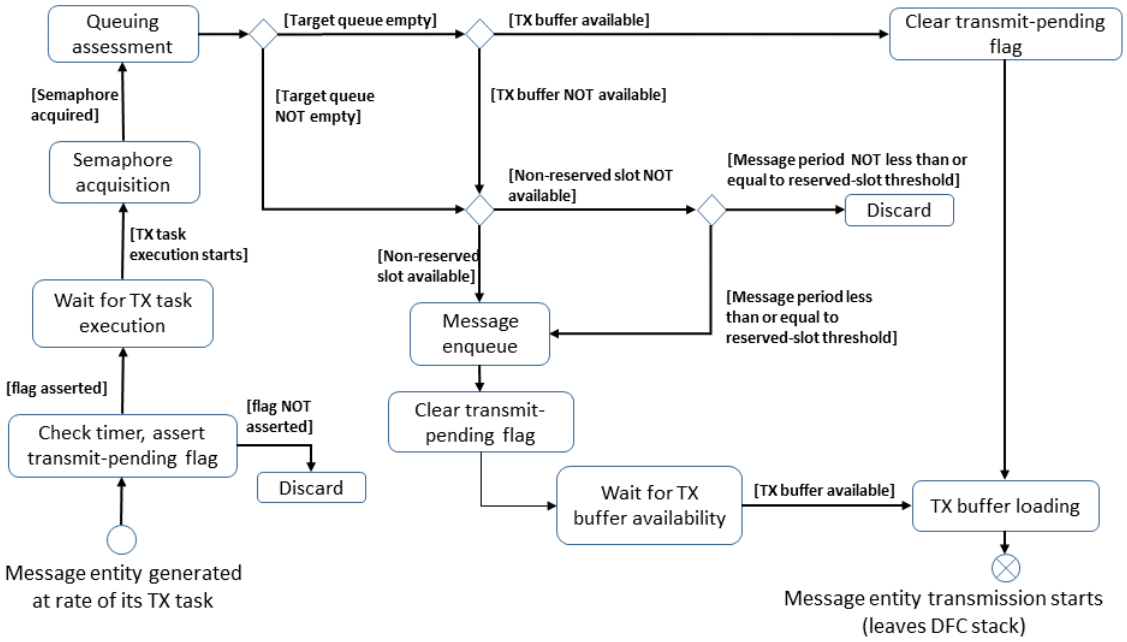

Figure 5.3: General Activity Flow of a Message in the DFC Stack

Each activity (in rectangular boxes) is associated with a time delay in the flow however a transition is not. The duration of the delays can vary for different messages

or different instances of the same message, since it is based on a number of real time factors inside the ECU. For instance, when a 25ms TX task message becomes pending for transmission, the duration of the "Wait for TX task execution" activity is the sum of the number of transmit-pending message in the 6.25ms and 12.5ms TX task and the inter-TX-task delays from the RTOS, as the 25ms TX task is the last to be executed among the 3 TX tasks. The duration of "Wait for TX buffer availability" activity varies based on the DLC of the message currently being transmitted and the current bus load. The duration of an activity can also be zero. In the cases of the 6.25ms TX task messages, whose TX task is the first to be executed in the DFC stack, the duration of the "Wait for TX task execution" activity is zero if there are no other tasks in the RTOS scheduled ahead of the 6.25ms TX task at its time of arrival.

The following sections will provide more details on the SimEvents features that are utilized in the model, and the major blocks that implement the behaviors of the DFC stack.

5.2.1 Global Data Store Memory Blocks

The Global Data Store Memory block allows declarations of data structures in a SimEvents model, and provides other MATLAB function blocks or SimEvents blocks in the same scope with access to the declared data structures.

There are three memory blocks deployed in the root level of the SimEvents model: 25ms TX task message timer, inter-transmission timer and RX message objects.

The 25ms TX task message timer memory block stores the index in message set, nominal transmit rate, current timer value and transmit-pending flag of the messages that are managed by the 25ms TX task. Since some of the 25ms TX task messages

become pending for transmission once every several instances of 25ms TX task, this memory block is necessary for keeping track of the values of the timers on these messages. Furthermore, in the case where a transmit-pending message is refused to enter a fully occupied queue, its transmit-pending is saved in the memory block so that it can be assessed again for queuing in the next instance of the 25ms TX task. In contrast, the 6.25ms and 12.5ms TX tasks transmit all their messages in each arrival, and therefore no memory blocks are required for the 6.25ms and 12.5ms TX tasks.

The inter-arrival time memory block keeps a timestamp for each message, which records the time of the previous successful transmission of each message ID on the CAN network (i.e. time of the most recent transmission-complete ISR event). Once a new instance of a message is successfully transmitted, the inter-arrival time can be calculated by taking the backward time difference between the current simulation time and the timestamp stored in the inter-arrival time memory block. Then the current simulation time will replace the old timestamp in the inter-arrival time memory block so it can be used in the calculation of next inter-arrival time.

The RX message objects memory block is used for basic reception functionalities of the simulation model, such as definition of messages ID to RX buffer mapping, RX message counter, *etc.*. It is part of the basic framework for message reception functionalities of the DFC stack which can be used for future expansion of this project.

5.2.2 Resource Management Subsystem Blocks

The Resource Management blocks allow custom resources to be defined and provides utilities to conveniently manage these resources. There are 3 types of Resource Management blocks in the SimEvents library: the Resource Pool block, the Resource

Acquire block and the Resource Release block.

Custom resources are defined in the Resource Pool blocks. Only 1 type of custom resource can be defined in each Resource Pool block, but it is possible to have multiple Resource Pool blocks in order to use a number of different types of custom resources in a SimEvents model. Aside from the name of the resource, each custom resource has properties including granularity, the amount of the resource and whether this resource is reusable upon release.

The Resource Acquire block allows an entered entity to request to acquire any amount of a single or multiple types of custom resources. Figure 5.4 shows all the configurable block parameters of a Resource Acquire block. When an entity enters, if the remaining amount of any requested resource is less than requested amount, the entity will be blocked and waits in the Resource Acquire block until at least the requested amount of resource becomes available. Once the entity departs from the block, the acquired amount of resource is subtracted from the remaining amount in the corresponding resource pool. The maximum number of waiting entities in a Resource Acquire block can also be specified as a configurable parameter. In the case where there are messages waiting to acquire resources inside multiple Resource Acquire blocks in a SimEvents model at the same time, SimEvents will first consider resource allocation for the Resources Acquire block with the highest acquisition priority.

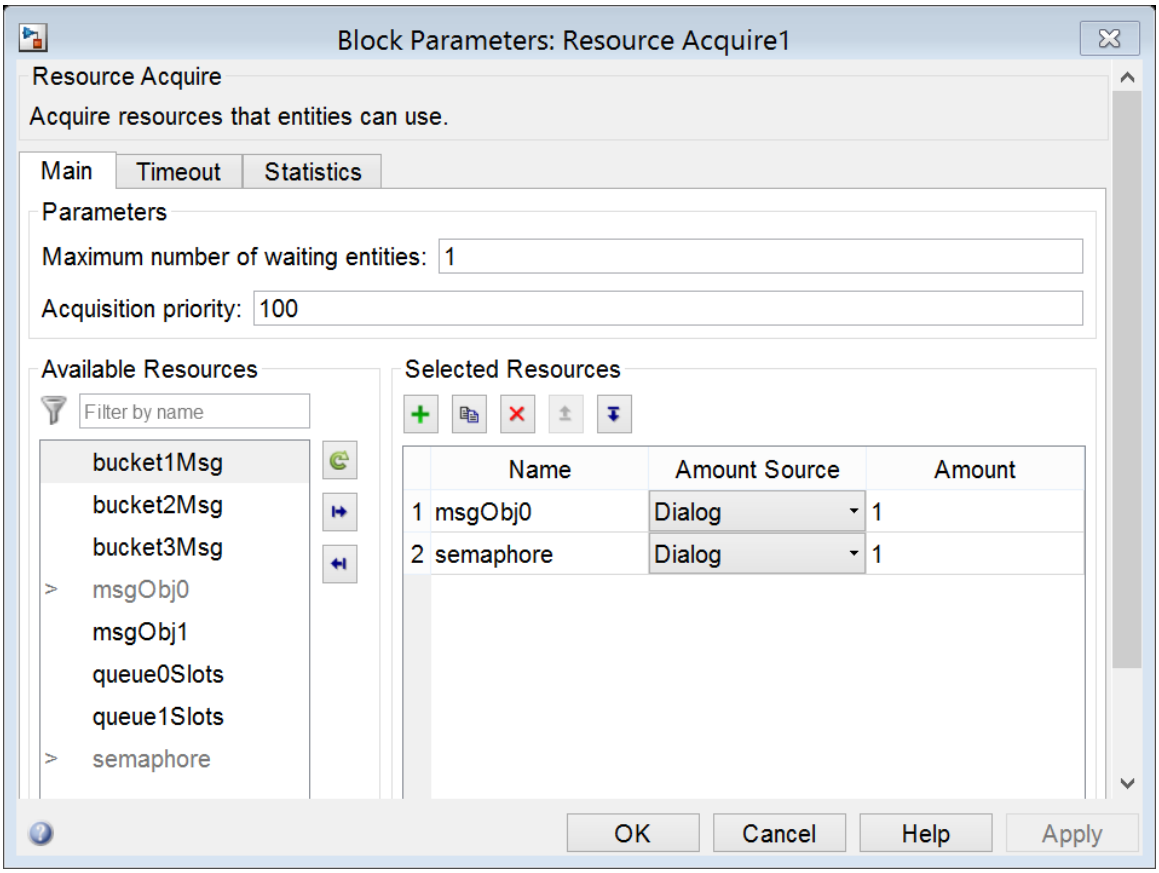

Figure 5.4: Configurable Parameters of Resource Acquire Block

Resources that are acquired by an entity will be attached to that entity until the entity enters a Resource Release block. The Resource Release blocks can be configured to release one or multiple types of resources that are attached to the entering entities. However once a type of resource is specified to be released, the entire amount of that resource attached to the entering entities will be released.

The Resource Management blocks provide utilities for conveniently managing the amounts of limited resources in the modeled system, and make it easier to enforce the priority of resource acquisition when multiple entities attempt to request resources at the same time, compared to using the Global Data Store Memory blocks to keep track

of the numbers of the resources and priorities of resource acquisition as implemented in the Vector CAN stack Whinton (2016) in the MATLAB 2014 environment.

In the DFC stack SimEvents model, several types of custom resources are simulated using the Resources Pool blocks. A TX buffer Resource Pool is defined for each transmit queue. Since there is only 1 TX buffer for each transmit queue in the DFC stack, the total amount of each of the TX buffer resource is set at 1 (however, the total amount of TX buffers is configurable and can be changed to other values if using the simulation model to simulate other software or hardware specifications). The TX buffer resource is meant to allow only 1 message to be "loaded in to the TX buffer" and make the rest of transmit-pending messages wait in the transmit queue. The semaphore, which protects the critical section of the queuing assessment, enqueue and dequeue events, is defined as a custom resource with total amount of 1 to ensure the mutual exclusion of the critical sections. The number of transmit-pending messages in each TX task is defined as a custom resource, so is the number of remaining available slots in each transmit FIFO queue, which is used to monitor the depth of queues. Although these are not technically resources in the actual system, they are defined as custom resources so that the model can utilize the Resource Management blocks for data manage during simulation or to control the advancement of message entities at certain places inside the model, such as making an entity waiting for a resources in a Resource Acquire block, *etc.*. More details will be provided in the following sections.

5.2.3 The Message Arrivals Subsystem Block

The Message Arrivals subsystem block manages the timing of new instance arrivals of the transmit messages as well as the generation of the message entities. It simulates parts of the TX task routines provided by the CAN Handler that manage the timers and transmit-pending flags of messages and prepares transmit-pending messages for queuing assessment.

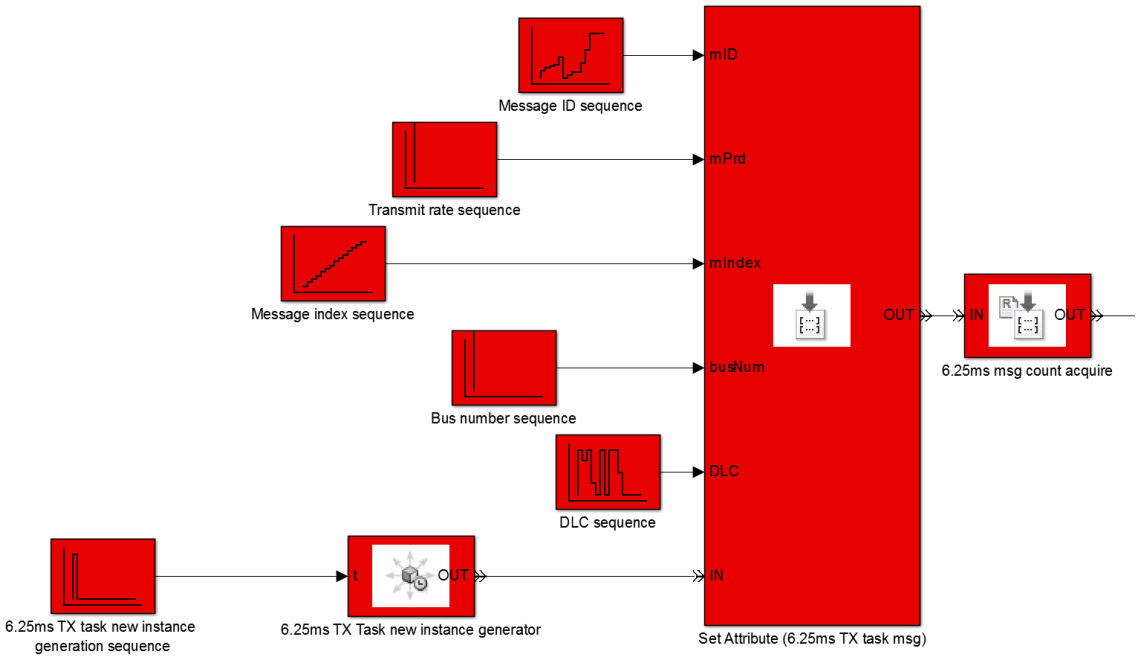

Figure 5.5: The Message Entity Generation Mechanism for the 6.25ms TX Task.

Figure 5.5 shows the mechanism for message entity generation for the 6.25ms TX task. Time-Based Entity Generator blocks are deployed to generate messages entities for the TX tasks. Since the TX tasks each have a fixed arrival rate and fixed number of periodic transmit messages, the generator block for each TX task is configured to repeatedly generate the same number of entities as its transmit messages at the frequency of its rate of arrival. The generated entities are blank entities, and, after

their departure from the generator block, they enter a Set Attribute block, which attaches the required information fir the transmit messages to them as entitie attributes. Each departed entity carries the information of a transmit message, which includes the message ID, nominal transmit rate, DLC, index in the message set (internal transmit priority) and bus number. After obtaining the message information, each entity will acquire a "transmit-pending message count resource", which is defined separately for each TX task. These message count resources indicate the number of remaining transmit-pending messages for each TX task that still need to be assessed for queuing, and are used for regulating the advancement of message entities in lower-priority TX tasks, which will be discussed in more details later in this section. This message entity genration mechanism is also deployed separately for the 12.5ms and 25ms TX tasks. Thus there are a total of 3 streams of messages entities generated in the Message Arrivals block, as shown in Figure 5.6. The red-colored blocks, yellow-colored blocks and the blue-colored blocks belong to the 6.25ms, 12.5ms and 25ms TX tasks, respectively.

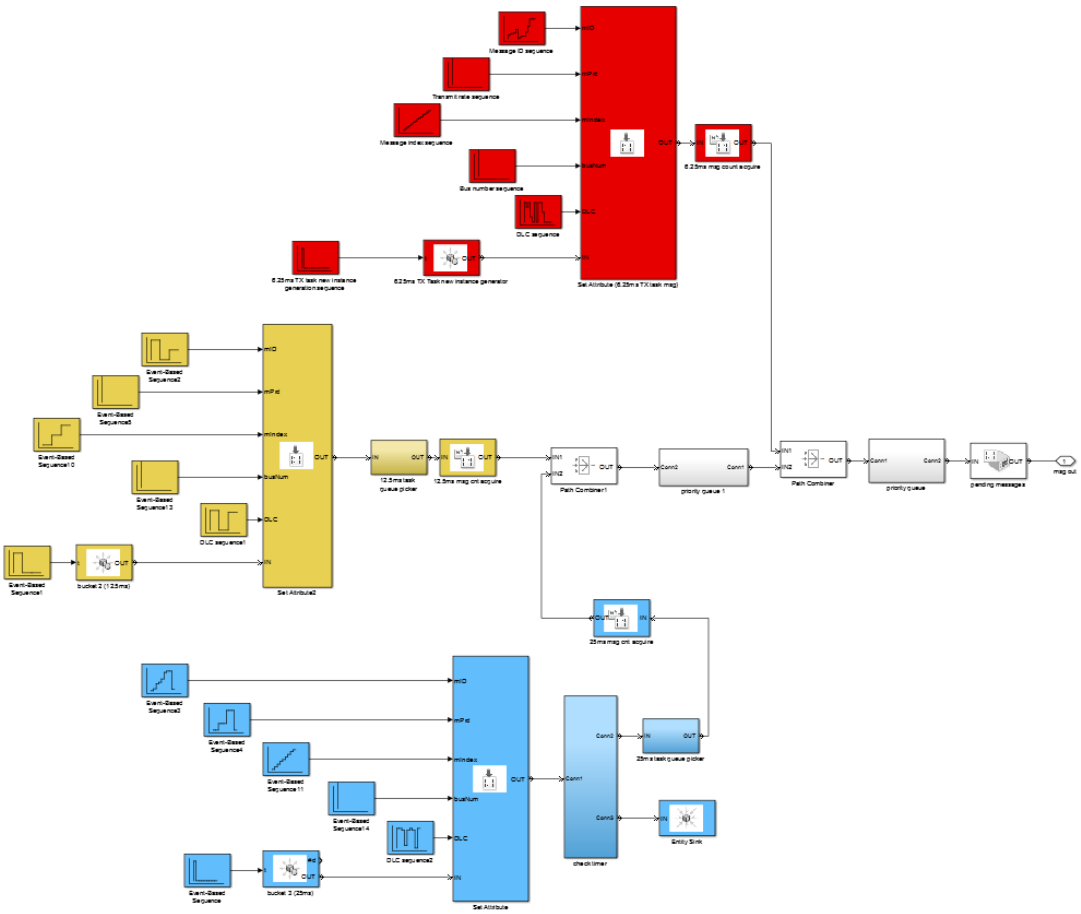

Figure 5.6: The Structure inside the Message Arrivals Block.

The Set Attribute block in the message entity generation mechanism for each TX task attaches message information to the entered blank entities in the order of descending transmit priority, so that, after the entities depart from the Set Attribute block, they advance in the order of their priorities. To further ensure the priorities of the messages once the message entities from all 3 streams merge, a Priority Queue subsystem block (implemented in the Vector CAN stack in Whinton (2016)) is used before the entities leave the Message Arrival subsystem block.

The message entity generation mechanisms are identical for the 6.25ms and the

12.5ms TX tasks, as all messages managed by these TX tasks become pending for transmission in every instance of their TX tasks. In contrast, the 25ms TX task handles messages that have slower transmit rates than 25ms, and it may take more than 1 instance of the 25ms TX task to increment the timers of these messages before they become pending for transmission. Therefore, all message entities generated in the 6.25ms and 12.5ms TX task routes are set out to leave the Message Arrivals block (and advance into queuing and transmission later), while the message entities generated in the 25ms TX task route, before they leave the Message Arrival block, will advance through a mechanism that decides if the timer of the entered message has expired. The timer-checking mechanism is contained in a Subsystem block labeled "Check timer", and its internal structure is shown in Figure 5.7. The center component of the mechanism is a MATLAB Function block (inside the Function-Call Subsystem block labeled "Update timer and flag") that updates the timer and checks the transmit-pending flag (saved in the 25ms TX task message timer memory block) of a passing message. If the timer expires or the flag is already asserted (if the message was transmit-pending in the previous instance of TX task but was not able to enter the transmit queue), the message will be able to leave the 25ms TX task route as a transmit-pending message, otherwise the message entity is discarded since that message is not yet supposed to become pending for transmission. In other words, even though an entity is generated for every transmit message in the 25ms TX task at the rate of once every 25ms, the generated entity gets to leave the Message Arrivals block and proceed further only if its corresponding message is pending for transmission at the time of its generation. Due to this nature, a 25ms TX task message entity will

not acquire the transmit-pending message count until it is confirmed to be transmit-pending by the timer-checking mechanism.

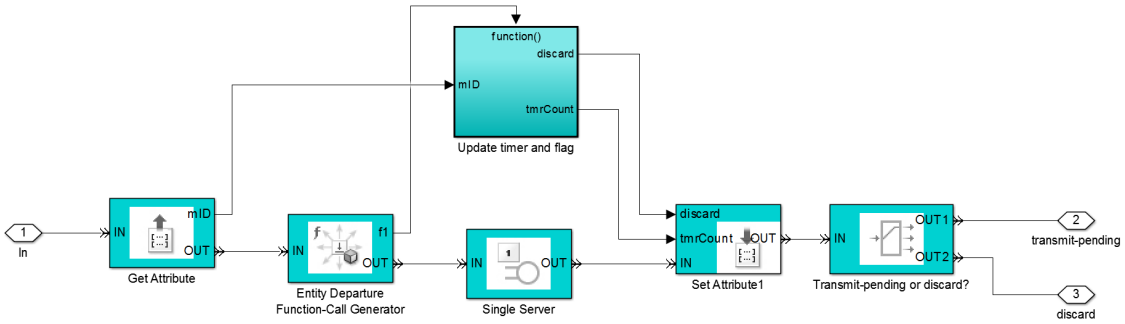

Figure 5.7: The Mechanism that Updates timer and Decides if a Message is Pending for Transmission.

Since the transmit messages from the 12.5ms and the 25ms TX tasks can be configured to transmit through either Queue 0 or Queue 1 based on the DFC stack calibration *DFC_Queue0_TxThresh*, a Queue Picker subsystem block is deployed in each of the 12.5ms and 25ms TX task message routes, which attaches a Queue Number attribute to each entered entity to indicate their target transmit FIFO queue before they leave the Message Arrivals subsystem block. The Queue Number attribute will later be used for determining the route of each message entity in the Queuing and Transmission block. Since the 6.25ms TX task messages are to be transmitted only through Queue 0, the Queue Number attribute of their entities always indicates Queue 0, and is attached by the Set Attribute block in the message entity generation mechanism of the 6.25ms TX task.

5.2.4 The Queuing and Transmission Subsystem Block

The Queuing and Transmission subsystem block contains the transmit message queues, their corresponding TX buffers and other SimEvents blocks that implement the queuing assessment, simulation data logging and part of the timing effects from the host environment.

The Queuing and Transmission block has 6 input ports and 1 output port, as shown previously in Figure 5.2. Besides the ports that allow message entities to flow in and out ("Msg in", "Msg out" and "TX-complete ISR" ports), the rest of the ports are connected to the "Amount in use" port of several Resource Pool blocks. The TX buffer of each transmit queue is defined as a reusable custom resource (previously acquired resources becomes available again upon release from an entity) that has a total amount of 1. The TX buffer resource will be acquired and held by a message entity throughout the duration while the simulated message is "loaded into the TX buffer and being transmitted on the bus". During this time, a value of "1" will be shown on the "Amount in use" port of the Resource Pool block of the TX buffer, which indicates that the TX buffer is busy so that no more message entities will be loaded into that buffer. The other 2 connected Resource Pool blocks define the "transmit-pending message count" resources for the 6.25ms and 12.5ms TX tasks.

The message input port of the Queuing and Transmission block is connected to the message output port of the Message Arrivals block. Once the message entities enter the Queuing and Transmission block, they are first separated and routed into 3 routes, each of which represents the route of a TX task, as shown in Figure 5.8 (there is an Attribute Scope block in each route that can display the passing message IDs with respect to time, for debug purposes). This structure is implemented in order

to simulate the inter-TX-task delays, the t_i's, which are applied before the queuing assessment of the first message of the 12.5ms and the 25ms TX tasks.

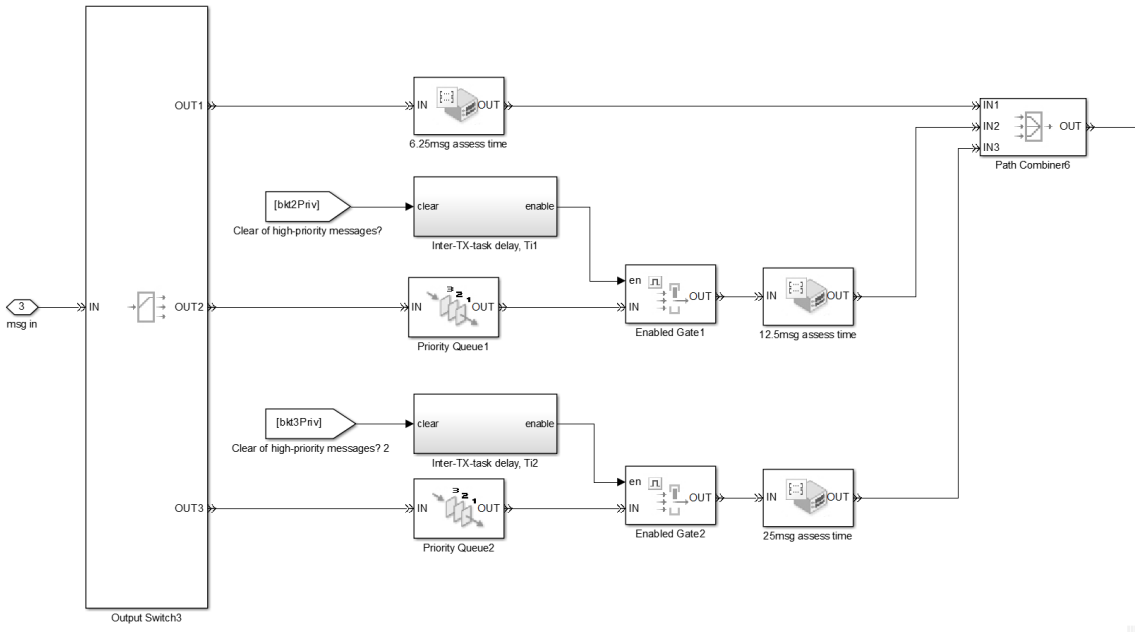

Figure 5.8: Mechanism that Divides Transmit-Pending Messages and Simulates Inter-TX-Task Delays.

As mentioned in Section 4.1.1, as the DFC stack runs inside a host environment such as the test bench, which has an RTOS, it is normal that the RTOS schedules other higher-priority tasks in-between the TX tasks of the DFC stack, and thus delays are introduced in-between the execution of the TX tasks. These inter-TX-task delays exist between the 6.25ms and the 12.5ms TX tasks, and between the 12.5ms and the 25ms TX tasks, and thus a delay mechanism is deployed in each of the 12.5ms and 25ms TX task routes (the lower 2 routes shown in Figure 5.8). The delay mechanism consists of an Enabled Gate block and the Inter-TX-Task Delay subsystem block that generates the enable signal for the gate. During the inter-TX-task delay, or while there is a higher-priority task transmitting its messages, the delay

subsystem block outputs a logic low to close the gate so that the message entities are stalled and have to wait the required time in the Priority Queue block before passing through the gate. Once it is the inter-TX-task delay is over, the gate is opened by a logic high signal from the delay subsystem block and the message in the Priority Queue block can then advance. The Inter-TX-Task Delay subsystem blocks output their enable signals based on the number of transmit-pending messages from higher-priority TX tasks (defined as custom resources in root level of the SimEvents model). A zero number of transmit-pending messages in higher-priority TX tasks indicate the completion of these TX tasks. Figure 5.9 shows the mechanism that converts the number of transmit-pending messages to the input signals for the Inter-TX-Task Delay blocks. For the 12.5ms TX task, when there is no transmit-pending messages from the 6.25ms TX task, the Compare To Zero block outputs a logic high (value "1") to a connected SimEvents Goto block, which sends the signal "wirelessly" to the Inter-TX-Task Delay block in the route of the 12.5ms TX task. For the 25ms TX task, since it is executed after both the 6.25ms and the 12.5ms TX tasks, the input signal to the delay block in its route is based on the number of transmit-pending messages of both the 6.25ms and 12.5ms TX tasks.

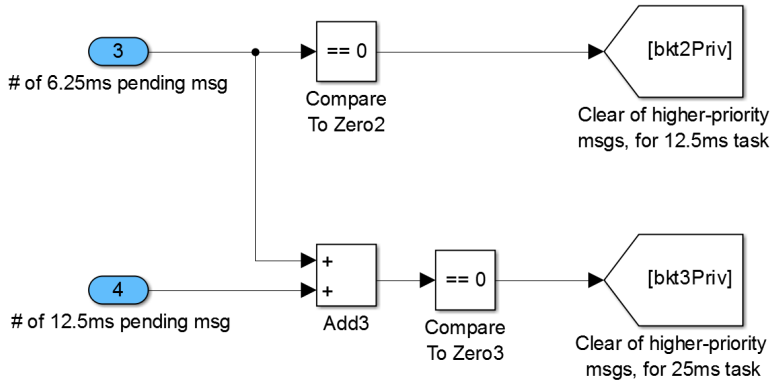

Figure 5.9: Signal Conversion from the Number of Transmit-Pending Messages to the Inputs for the Inter-TX-Task Delay Subsystem Blocks.

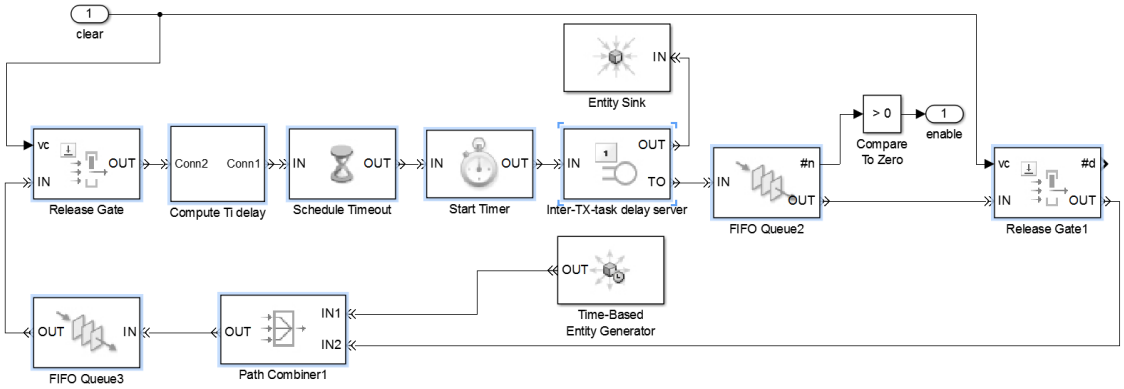

Figure 5.10: The Internal Implementation of the Inter-TX-Task Delay Subsystem Block.

The internal implementation of the Inter-TX-Task Delay subsystem block, as illustrated in Figure 5.10, is a closed loop (high-lighted blocks) structure that generates the enable signal by using the flow of a single SimEvents entity. The Time-based Entity Generator generates an entity at the start of the simulation and sends it into the closed loop from the top input port of the Path Combiner Block. The entity will then wait in a FIFO Queue block until the Release Gate blocks opens when the input

signal to the delay block shows a rising edge (changes from logic low to logic high, completion of higher-priority TX tasks). After the Release Gate block, the Compute Ti Delay subsystem block will generate and attach a duration of inter-TX-task delay (based on the specified modeling option for inter-TX-task delay in the simulation script) to the entity as an attribute, and the duration is then set as the Timeout duration of the entity by the Schedule Timeout block. The Start Timer block then starts the Timeout timer on the entity before the entity enters and waits in the Single Server block labeled "Inter-TX-task delay server". The server block has an infinite service time, which would keep the entered entity inside indefinitely. However, when the Timeout timer of the entity expires the TO port (Timeout port) of the server block is enabled, which provides the entity in this subsystem an exit. As a result, the entity stays in the Single Server block for exactly the duration of the attached inter-TX-task delay. Once it leaves the server block, the entity enters a second FIFO Queue block. While the entity stays in the queue, the Number of Entities in Queue signal port (labeled "#n") becomes "1" and will thus gives a logic high to the output of the Inter-TX-Task Delay subsystem block. The output will stay high until the second Release Gate block opens and allows the entity to leave the FIFO Queue block. The second release gate opens on a falling edge from the input signal of the Inter-TX-Task delay block, which indicates the start of a new instance of any higher-priority TX task. The entity will then again enter the first FIFO Queue block at the start of the closed loop and wait for the completion of the higher-priority tasks.

The Compute Ti Delay subsystem block implements the available modeling options for the inter-TX-task delay. Figure 5.11 below shows the internal structure of the subsystem block. An entity that enters the subsystem block is first routed by an

Output Switch block to one of 5 routes, each of which corresponds to one of the 5 modeling options. From top to the bottom of the entity routes, the delay durations are generated based on a constant value, normal distribution, gamma distribution, random value from the actual delay data from the test bench results and kernel distribution, respectively, which cover modeling options of using both deterministic models (constant values) and statistical models (parametric and non-parametric probability distribution functions, more details on using the statistical models provided in Chapter 6). The top route, which attaches a constant value to the inter-TX-task delay, uses a single Set Attribute block and simply attaches the constant value specified in the simulation script to the entity. The rest of the routes have similar structures: a pseudo-random inter-TX-task delay duration is generated from a Event-Based Random Number block and the delay duration value is then passed into a mechanism that constrains that value within the bounds of inter-TX-task profiled from the test bench traces.

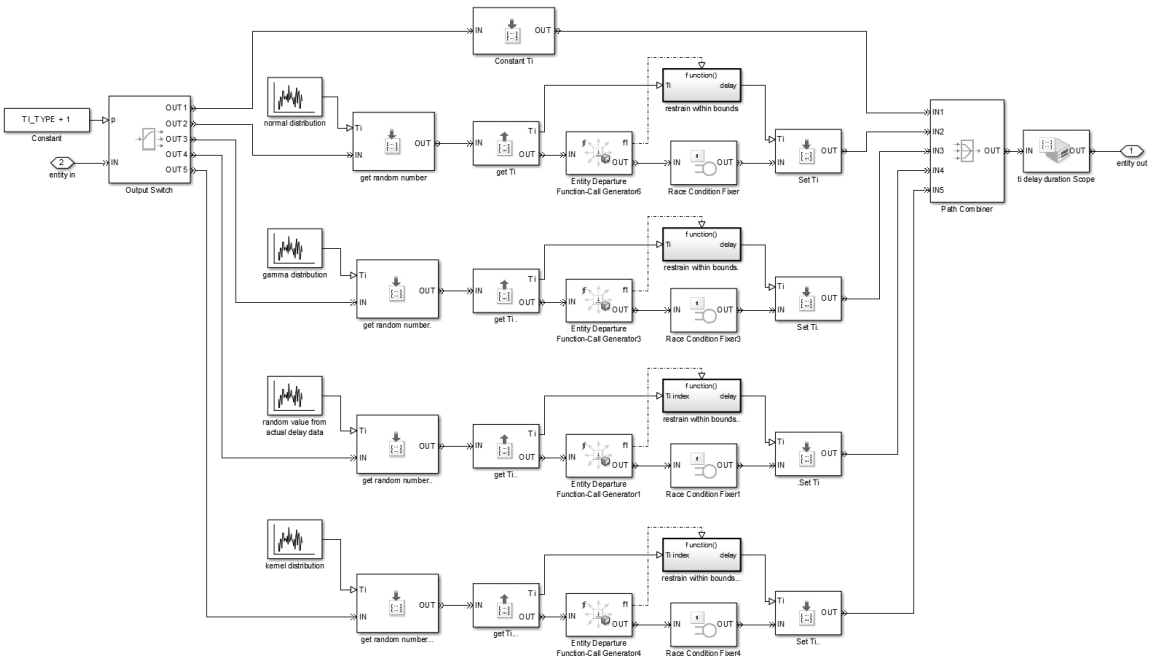

Figure 5.11: The Internal Implementation of the Compute Ti Delay Subsystem Block.

Back to the Queuing and Transmission subsystem block, once the message entities leave the route of its TX task and advance past the Path Combiner block in Figure 5.8, they enter a Priority Queue block and wait for the queuing assessment in the order of their internal transmit priorities, as shown in Figure 5.12. Their arrival at the Priority Queue block indicates the start of execution of their TX task. The message entities will then one by one advance into the "Semaphore acquire" Resource Acquire block to request the semaphore resource that protects the critical sections of the queuing assessment, enqueue and dequeue events, which access the contents of the transmit queues and TX buffers. The semaphore resource pool has the amount of 1, which means that only 1 message entity will be able to acquire the semaphore at any given time and thus enforces the mutual exclusion of these events. Beside the semaphore resource, another custom resource is acquired when an entity leaves the

Resource Acquire block: the Enqueue Core resource. The Enqueue Core resource is custom resource defined in the scope of the Queuing and Transmission subsystem block. It is is meant to be acquired by an message entity that is being processed by the enqueue core of the dual-core MCU, i.e. during the queuing assessment and the enqueue process. The Enqueue Core resource is also used for detecting an irregular enqueue event, which will be discussed in more details later in this section.

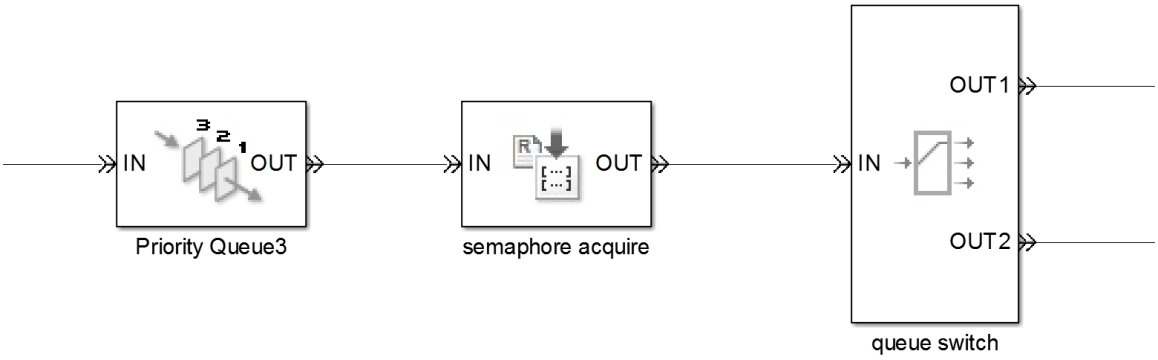

Figure 5.12: Acquisition of Semaphore before Queuing Assessment

After the resource acquisition, the entity of a transmit-pending message enters one of the two routes that model the two transmit FIFO queues (as well as their corresponding TX buffers) of the DFC stack. The routing of message entities is performed by an Output Switch block based on the Queue Number attribute that is attached to each messages entity in the Message Arrivals subsystem block. Figure 5.13 shows the SimEvents structure inside the Queuing and Transmission block that implements the entity routes for the 2 transmit FIFO queues. The green-background blocks belong to the route for Queue 0 and the blue-background blocks belong to Queue 1. The orange-background blocks implement the simulation data logging mechanism, which will be introduced in the sections below.

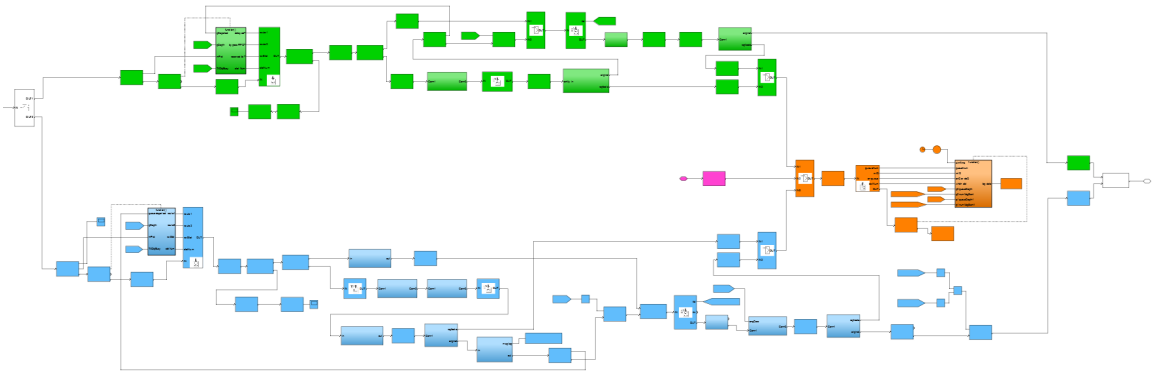

Figure 5.13: SimEvents Blocks that Implement the Transmit Queues, the TX Buffers and Simulation Data Logging mechanism.

Aside from the common characteristics between the 2 transmit FIFO queues of the DFC stack, more features have been implemented in the SimEvents model for Queue 1 compared to Queue 0 (as shown by the difference in complexity between the blue-background route and the green-background route in Figure 5.13). The extra features include timer management of the 25ms TX messages, modeling of the irregular enqueue times and data logging for the reserved slots. For better explanation of the transmit FIFO queue related implementation, the structures in the green-background message entity route for Queue 0, which is common to both queues, will be discussed first, and the extra structures in the Queue 1 route will be discussed afterwards.

Once a transmit-pending message entity enters the route for Queue 0, it will advance through a mechanism as shown in Figure 5.14, where the queuing assessment of message entities is performed.

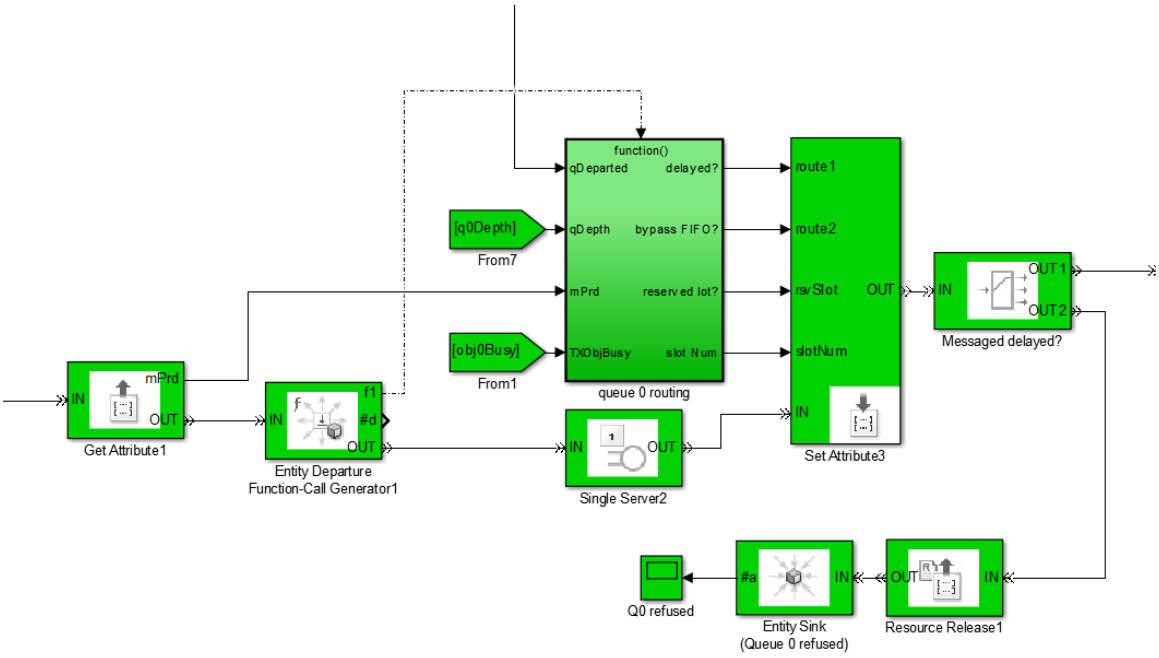

Figure 5.14: The Queuing Assessment Mechanism.

The decision for the queuing assessment is made in the Function-Call Subsystem Block in the center of the mechanism labeled "Queue 0 Routing". It collects the necessary message information from the message entity as well as real-time information of the DFC stack including the depth of Queue 0, number of messages departed from Queue 0 (used to decide the slot number in the ring buffer for the next incoming message) and TX buffer availability, and then makes the decision whether the entered message is eligible for transmission (enter the queue or bypass the queue and enter the TX buffer). The decision of the queuing assessment of any message entity is attached as attributes, which will be used by an Output Switch (labeled "Message delayed?") that either routes the message entity into an entity sink (not eligible for transmission, therefore discarded) or lets the entity advance to Queue 0 or the TX buffer (eligible

for transmission) through the "OUT1" port. The nominal transmit rate of a message (abbreviated as "mPrd" in the model, or message period) is extracted from the message entity, and is used to decide whether the message is eligible for entering a reserved slot when all unreserved slots are occupied. After the assessment decision has been made, the "rsvSlot" attribute, which indicates whether the assessed message is placed into a reserved slot, is attached to the entity for simulation data logging purposes, and so is the number of the slot in the FIFO queue that the message is placed in.

For a message that is eligible for transmission, which departs the "Message Delayed?" Output Switch block from the "OUT1" port, it will propagate through one of the 2 routes of the transmit queue mechanism shown in Figure 5.15.

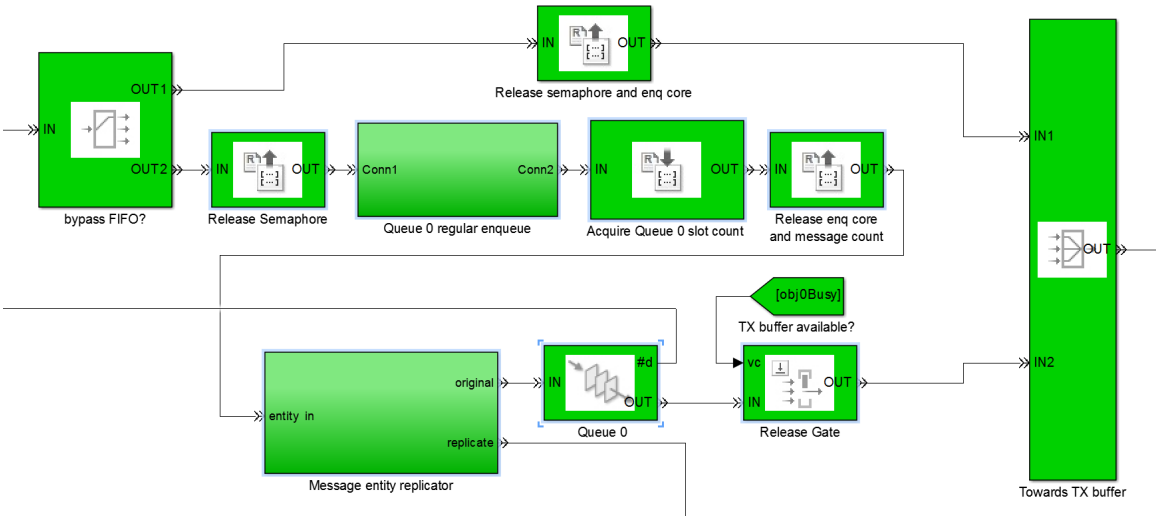

Figure 5.15: The Transmit Queue (Queue 0) Mechanism.

The top route in the mechanism is for messages that bypass the FIFO queue and enter the TX buffer directly. The only SimEvents block in the top route is a Resource

Release block that releases the semaphore and enqueue core resources from the departed message entity, since the queuing assessment of the entity has completed. The bottom route implements the queue process of the messages. An entering message entity first releases only the semaphore resources at the "Release semaphore" Resource Release block and is then ready to be loaded into Queue 0. The duration of the enqueue is simulated in the Queue 0 Regular Enqueue subsystem block, which generates a duration of the regular enqueue time based on the modeling option of the regular enqueue time specified in the simulation scripts, and stalls the entered message entity for the generated time duration. The Queue 0 Regular Enqueue subsystem block has a similar internal structure as Compute Ti Delay subsystem block as shown earlier in Figure 5.11, in order to implement multiple modeling options for generating the regular enqueue times. On top of the common structures, there is an extra Single Server block before the exit of the subsystem block that stalls the entered entity for the generated regular enqueue time duration. Once a message entity departs from the Queue 0 Regular Enqueue block, it is considered "inside Queue 0", and it will immediately acquire a queue 0 slot count resource (used to keep track of the number of queued messages in Queue 0) and release the enqueue core and transmit-pending message count resources. Before it enters the FIFO Queue block labeled "Queue 0", which is the actual FIFO queue that holds the enqueue Queue 0 messages, it needs to go through the Message Entity Replicator subsystem block. The Message Entity Replicator block creates a replica of the entered entity that carries the same types and values of attributes, and then sends the replica to the simulation data logging mechanism, which allows the data logging mechanism to extract the attached attributes from the the replica and record the enqueue event (more details provided later in

Section 5.2.6). The SimEvents library provides a Replicate block but it is not used in the simulation model due to the fact that the message entities in the simulation model carry custom resources from time to time and resources can not be replicated. Instead, the Message Entity Replicator subsystem is implemented to generate replicas of the entered message and the replicas possesses only the attributes of the original entity but not the custom resources. Finally, for message entities that wait in the "Queue 0" Priority Queue block after the Message Entity Replicator block, whenever the TX buffer becomes available (informed wirelessly by the SimEvents From block that transmits the number of in-use Queue 0 TX buffer resource), the message entity at the head Queue 0 will advance forward to the TX buffer from the "Towards TX buffer" Path Combiner block.

On the other sided of the "Towards TX Buffer" Path Combiner block, is the mechanism that implements the TX buffer. The structure of the TX buffer mechanism is shown in Figure 5.16. When a message entity enters, it will first request the TX buffer and semaphore resources in the Resource Acquire block. The Number of Entities Departed port (labeled "#d") of the Resource Acquire block is enabled to send the number of transmitted Queue 0 messages to the queuing assessment mechanism wirelessly through a Goto block. Once it has acquired the resources, the message entity will enter the TX Buffer Loading Time subsystem block, which stalls the entity in order to simulate the duration it takes to load the message into the TX buffer. In the case of this thesis, the TX buffer loading time is zero for all simulations conducted on the simulation model, however, this block provides modeling options in buffer loading times that can be used for future work with the simulation model. After the TX buffer loading of the message, the semaphore and Queue 0 slot count resources

are released from the entity. From this point, the message entity only carries the TX buffer resources, since it is in the state of "being transmitted on the CAN bus in the TX buffer". The entity then is ready to leave the Queuing and Transmission block and "start its transmission on the CAN network" (leave the DFC stack SimEvents model and enter the CAN network) after passing through a Message Entity Replicator block, which logs the dequeue event (or, TX buffer loading event).

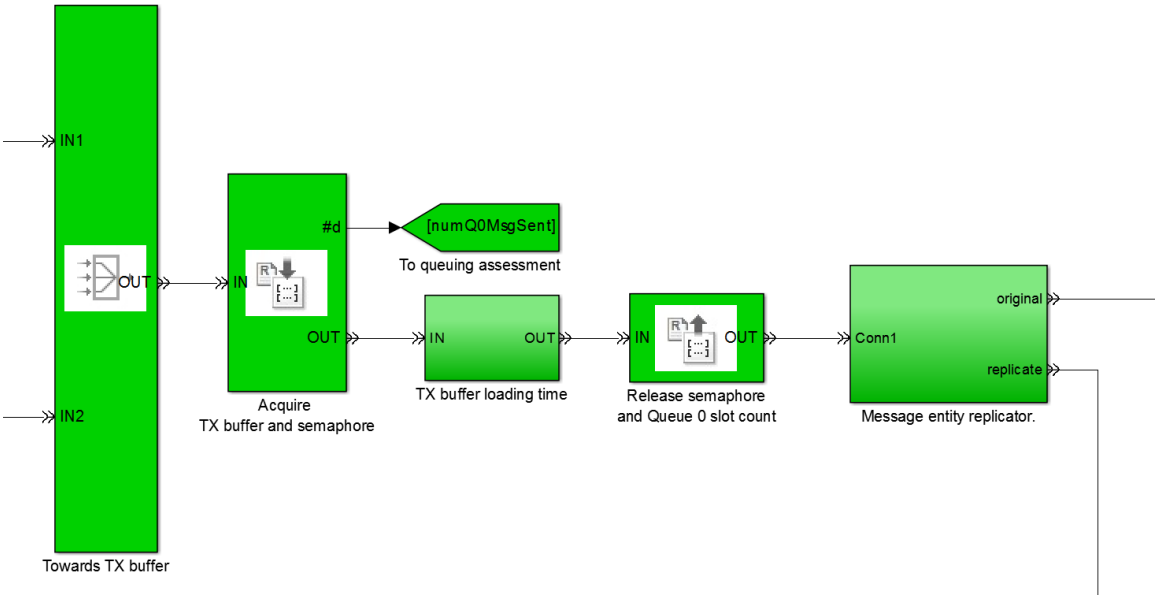

Figure 5.16: The TX Buffer Mechanism.

The output of the TX buffer mechanism (connected to the "Original" output port of the Message Entity Replicator block) of the Queue 0 route is combined with the output of the TX buffer mechanism of the Queue 1 route through a Path Combiner block, so that the message entities that are transmitted through both of these routes leave the Queuing and Transmission subsystem block through a common output port.

For the Queue 1 route, the extra features includes subsystem blocks that handle the timing of the 25ms TX messages, the irregular enqueue times and logging of usage

of the reserved slots.

The 25ms TX Message Timing Management subsystem block is created to manage the timers and transmit-pending flags of the 25ms TX task messages, and its internal implementation is shown in Figure 5.17.

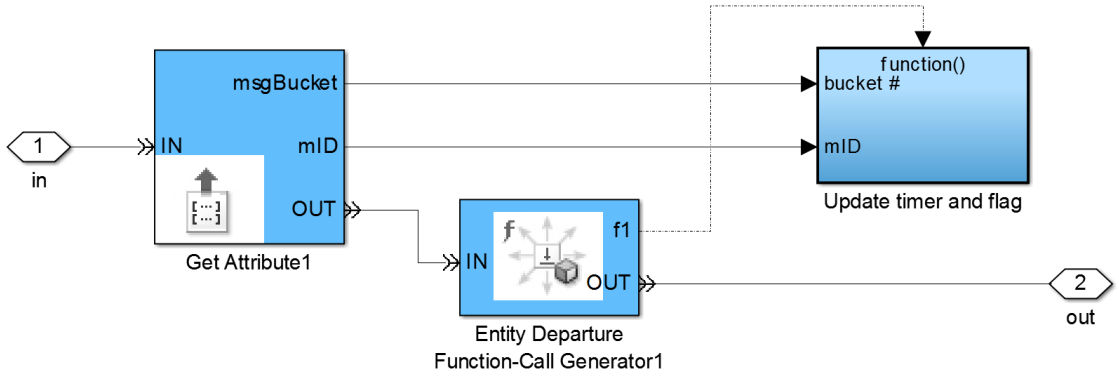

Figure 5.17: The Internal Implementation of the 25ms TX Message Timing Management Subsystem Block.

An entered message entity first arrives at a Get Attribute block and provides its message ID and a number that identifies its TX task. The message ID and the bucket number (indicates the TX task that the message belongs to) are then used as inputs to the "Update Timer and Flag" Function-Call Subsystem block, which clears the corresponding timer and transmit-pending flag of the entered message saved in the 25ms TX task message timer memory block. Checking the bucket number is necessary, since even though both messages from the 12.5ms and 25ms TX tasks are transmitted through Queue 1, only the 25ms TX task messages have timers.

This 25ms TX Message Timing Management block is deployed in both of the routes of the transmit queue mechanism of Queue 1, so that when a 25ms TX task message is eligible for transmission, whether bypass or enter the queue, its timer and

transmit-pending flag are reset.

In order to simulated the irregular enqueue times, 2 Subsystem blocks are implemented. First, the triggering condition for a irregular enqueue event needs to be detected, which is a dequeue event of a queued message that happens within the duration of the regular enqueue process of a transmit-pending message. The Irregular Enqueue Detection subsystem block is deployed in the TX buffer mechanism of Queue 1 after the TX Buffer Loading Time block, and its internal structure is shown in Figure 5.18. When a message enters this subsystem block, at which point it is considered "already loaded into the TX buffer", its entry will generate a function call to the MATLAB Function block that checks the availability of the enqueue core resource. If the enqueue core is being used, which indicates an on-going enqueue process at the time, then the irregular enqueue triggering condition is detected and the duration of the on-going enqueue process should be extended. The result of detection from the Irregular Enqueue Detection subsystem block is stored in a Global Data Store Memory block.

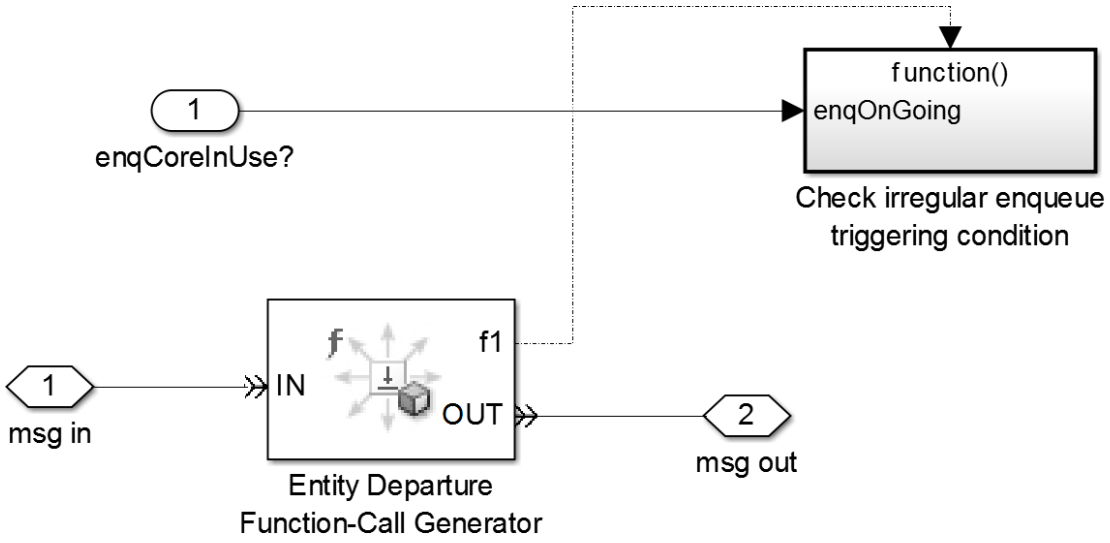

Figure 5.18: The Internal Implementation of the Irregular Enqueue Detection Subsystem Block.

Once an irregular enqueue triggering condition has been detected, the duration of the enqueue process is extended by the Enqueue Time Adjustment subsystem block, which is deployed after the Regular Enqueue subsystem block in the second route of the transmit queue mechanism of Queue 1. The internal structure is shown in Figure 5.19. The entry of a message entity will first generate a function call to the "Irregular Enqueue Detected?" Function-Call subsystem block to check the result of triggering condition detection stored in the Global Data Store Memory block. If no irregular enqueue has been detected, the message entity will advance through the "OUT1" port to the top route and leave the Irregular Enqueue Detection subsystem block without any time delay. If an irregular enqueue condition has been detected, the message entity will be routed to the mechanism shown in the bottom half of the figure, which is very similar to the internal structure in the Computer Ti Delay subsystem block and the Queue 0 Regular Enqueue subsystem block mentioned before.

The mechanism generates the duration of the irregular enqueue time according to the specified modeling option (in the simulation scripts) for the irregular enqueue times, and then delays the message entity for the computed duration to simulated the irregular enqueue.

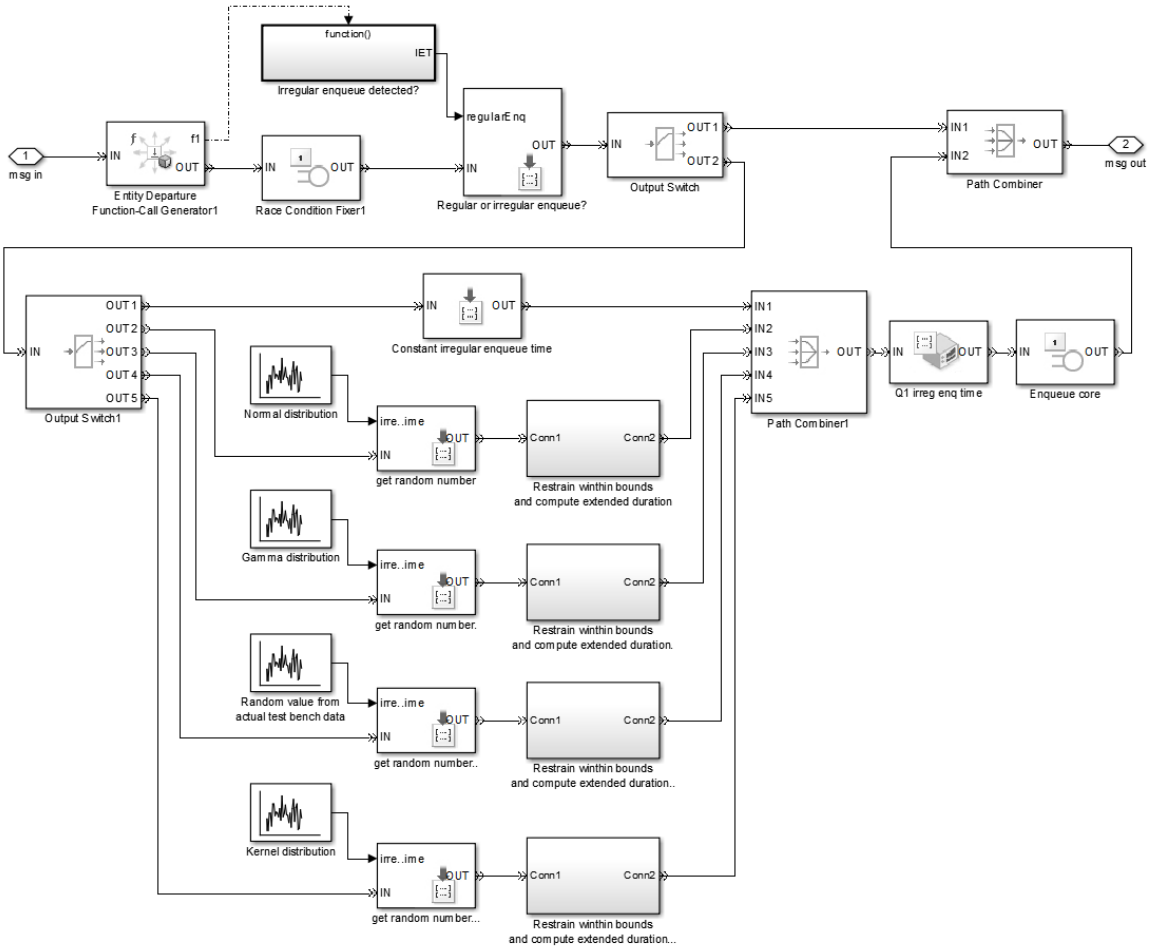

Figure 5.19: The Internal Implementation of the Enqueue Time Adjustment Subsystem Block.

The last of the extra features in the Queue 1 route is the Reserved Slot Logging subsystem block, which records of list of timestamps of messages that are loaded into the reserved slots in Queue 1. The internal structure is shown in Figure 5.20. The

block is deployed in the transmit queue mechanism of the Queue 1 route after the Regular Enqueue and Enqueue Time Adjustment subsystem blocks, which means all enqueue message entities will advance pass this block. The entered message entity is routed to either the top route, if the message is loaded into a reserved slot, or the bottom route, if it is loaded into a non-reserved slot, based on the "rsvSlot" attribute attached after the queuing assessment. If a message entity is loaded into a non-reserved slot, the entity simply exits the subsystem block. If the message entity is loaded into a reserved slot, the current simulation time (from a Clock block) is first attached to the entity through the Set Attribute block, and is then extracted and saved to a MATLAB workspace log together with the message ID and slot number of the enqueued message.

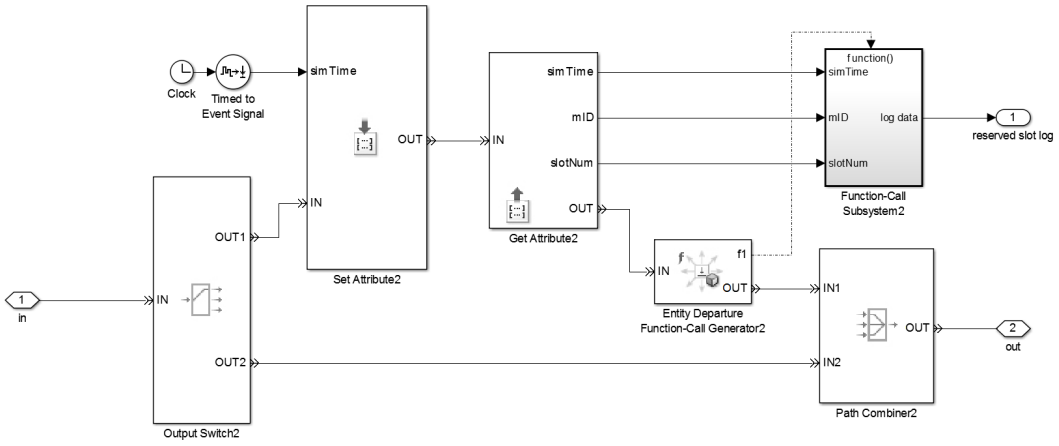

Figure 5.20: The Internal Implementation of the Reserved Slot Logging Subsystem Block.

5.2.5 Return of Successfully Transmitted Messages

Any message that is originally transmitted by the DFC stack SimEvents model will be routed back into the model by the Vector CAN bus block (Whinton (2016)) through

the Return port. Figure 5.21 shows the route for a returned message entity in the root level of the SimEvents model before it re-enters the Queuing and Transmission subsystem block.

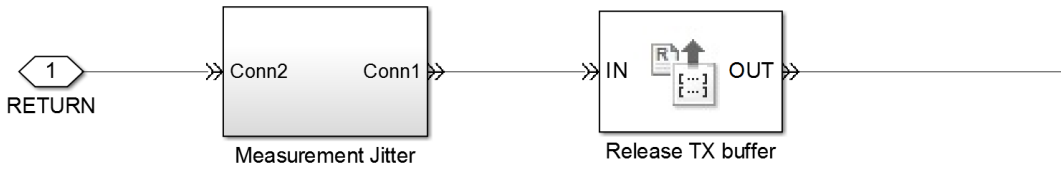

Figure 5.21: The Return Route for Successfully Transmitted Messages in Root Level of the SimEvents Model.

The Measurement Jitter subsystem block models the measurement jitter introduced by the host environment. It has a similar internal structure to other subsystems that are used to model the timing effects from the host environment, more specifically, the Compute Ti Delay subsystem, the Regular Enqueue subsystem and the Enqueue Time Adjustment subsystem blocks. The duration of the jitter is generated with one of the 5 modeling options (constant value, random value from the actual jitter data and pseudo-random number based on a normal distribution, gamma distribution or kernel distribution) based on the specified option in the simulation script.

Once the measurement jitter has been applied, the returned message entity enters a Resource Release block and releases the TX buffer resource it has been carrying from the start of its transmission (after it is loaded into the TX buffer). The released TX buffer resource can then be acquired by a queued message in the transmit queues if there is any.

After the release of the TX buffer resource, the message entity has finished the complete life cycle of its transmission. It will enter the Queuing and Transmission subsystem block and arrive at the simulation data logging mechanism so that the

transmission-complete ISR event can be captured. Differring from the way that the enqueue and dequeue events are logged through the use of a replica of the message entity, the logging for ISR events uses the original message entity as the entity at this point carries no resources and is to be discarded at a Entity Sink.

5.2.6 Simulation Data Logging

The simulation data logging mechanism is a structure implemented in the Queuing and Transmission subsystem block, shown as orange-background blocks in Figure 5.13. The mechanism captures the enqueue, dequeue and transmission-complete ISR events in the DFC stack simulation model, and saves the list of events in a MATLAB workspace variable, which is referred to as the "model generated trace".

As briefly mentioned in previous sections, the simulation data logging mechanism records events through receiving SimEvents entities that carry information of the event and of the message involved in that event ("event entities"). For enqueue or dequeue events, a replica of the enqueued or dequeued message entity is created and sent to the simulation data logging mechanism at the time of the event, carrying information such as the message ID, slot number in the queue, *etc.* Since there is zero elapsed simulation time between the creation of the replica and the reception of the replica at the simulation data logging mechanism, the exact time of the event can be captured. For the transmission-complete ISR events, the original message entity is received at the simulation data logging mechanism instead of a replica. As a results, a total of 5 input entity routes are connected to the simulation data logging mechanism, as shown in Figure 5.22, where the routes with blocks in green background are for the enqueue and dequeue events in Queue 0, respectively, routes

with blue-background blocks are for the enqueue and dequeue events in Queue 1 and the magenta-background route is for returned message entities of the transmission-complete ISR events. The Set Attribute blocks in the input entity routes attach an attribute that indicates the type of event, which can be the enqueue, the dequeue or the transmission-complete ISR event.

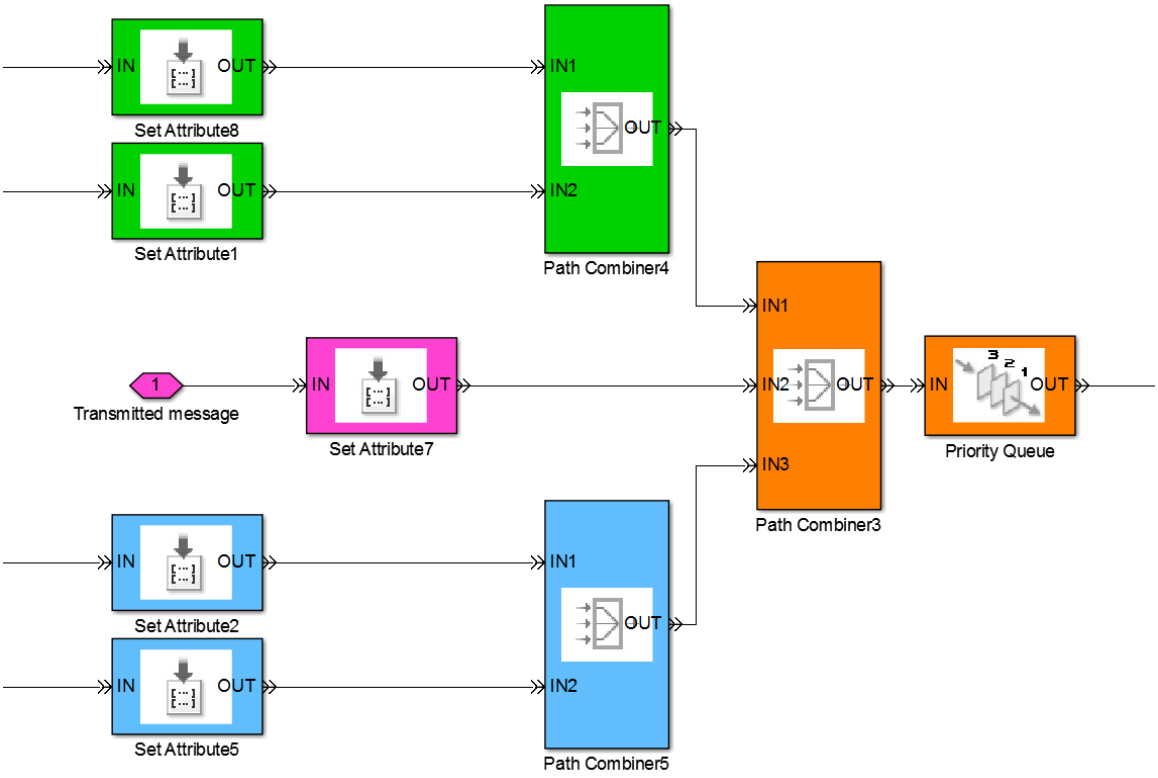

Figure 5.22: The Merge of Input Entity Routes before the Simulation Data Logging Mechanism.

A Priority Queue block is deployed after the Path Combiner blocks to sort the event entities that arrive at exactly the same time, based on the type of the event. The order of priority from high to low is: the transmit-complete ISR event, the dequeue event, the enqueue event. With TX buffer loading time set to zero, after

a transmission-complete ISR event, a message is dequeued and loaded into the TX buffer with zero elapsed simulation time, which will cause the event entity for the dequeue event to arrive at the logging mechanism at exactly the same time as the transmission-complete ISR event entity. In situations like this, the Priority Queue block will make sure that these 2 consecutive events are logged in the correct order by putting the transmission-complete ISR event entity at the head of the queue.

Once an event entity leaves the Priority Queue block, it enters the simulation data logging Mechanism, which is shown in Figure 5.23.

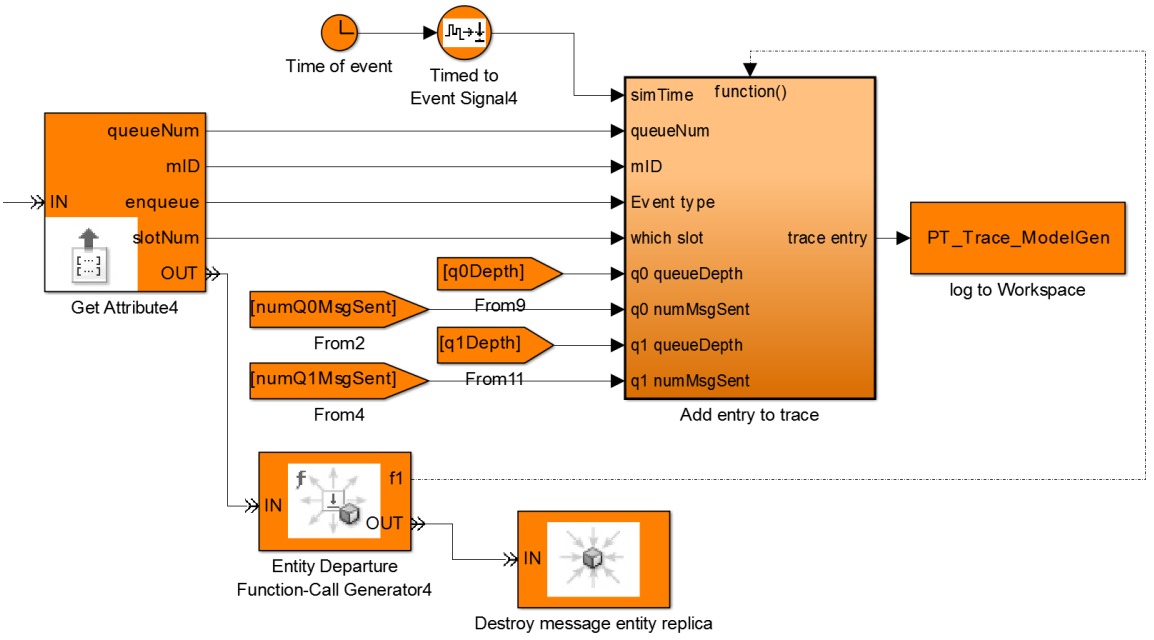

Figure 5.23: The Simulation Data Logging Mechanism.

The information of the captured event and associated message is extracted from the entered event entity through the Get Attribute block, and is then fed into a Function-Call Subsystem block together with the current simulation, queue depth and number of transmitted messages of both Queue 0 and Queue 1. Each time

an event entity enters the logging mechanism, the "Add Entry to Trace" Function-Call Subsystem block formats the received information into an entry for the model generated trace in the format of an array of values, and then outputs the array to the To Workspace block, which appends the new entry to the workspace variable of the trace. Any entity that enters the simulation data logging mechanism is discarded in an Entity Sink block after its associated event has been recorded.

Figure 5.24 shows a section of a raw model-generated trace from a simulation run. A "-1" is shown in the cells that are to be ignored, and it is used instead of zero because some of the useful values in the model generated trace can actually be zero. Similar to the parsed test bench trace, a row in the model generated trace represents a captured event. The first column shows the time of the event in milliseconds. The red-background columns show the message ID of either an enqueue or a dequeue event that takes place in Queue 0, if a message ID appears in these column. Similarly, the blue-background columns shows the message IDs associated with the enqueue or dequeue events in Queue 1. The message ID of a transmission-complete ISR event is shown in the green-background column. The last 3 columns on the right of the chart show the queue depth and total number of transmitted messages of the associated queue of the current event, and the backward time difference between the current and the previous rows, respectively.

	Queue 0		Queue 1					
Simulation time (ms)	Enqueue ID	Dequeue ID	Enqueue ID	Dequeue ID	ISR msg ID	Queue depth	Num transmitted msg	Backward time diff (ms)
91.3180	-1	-1	396	2	-1	1	31	0.0095
91.3260	-1	-1	397	3	-1	2	31	0.0078
91.5500	-1	-1	-1	-1	398	2	31	0.2235
91.5500	-1	-1	-1	396	-1	1	32	0.0000
91.6550	-1	-1	-1	-1	396	1	32	0.1053
91.6550	-1	-1	-1	397	-1	0	33	0.0000
91.8780	-1	-1	-1	-1	397	0	33	0.2226
93.7500	-1	170	-1	-1	-1	0	155	1.8724
93.7580	190	1	-1	-1	-1	1	155	0.0077
93.7640	201	2	-1	-1	-1	2	155	0.0064
93.7720	211	3	-1	-1	-1	3	155	0.0078
93.7770	249	4	-1	-1	-1	4	155	0.0052
93.7850	135	5	-1	-1	-1	5	155	0.0075
93.7920	145	6	-1	-1	-1	6	155	0.0069
93.7990	165	7	-1	-1	-1	7	155	0.0073
93.8050	167	8	-1	-1	-1	8	155	0.0062
93.8130	217	9	-1	-1	-1	9	155	0.0076
93.8210	290	10	-1	-1	-1	10	155	0.0079
93.9950	-1	-1	-1	-1	170	10	155	0.1741
93.9950	-1	190	-1	-1	-1	9	156	0.0000
94.2110	-1	-1	-1	-1	190	9	156	0.2162
94.2110	-1	201	-1	-1	-1	8	157	0.0000

Figure 5.24: A Section of a Sample Model Generated Trace.

Chapter 6

Results and Validation

This section will discuss the results from the simulations conducted on the DFC stack simulation model and validate the accuracy of the simulation results against the test bench experiment results.

Similar to the test bench experiments, the DFC stack will be the only transmitting node on the CAN network, which means there will be no other sources of bus load on the CAN network. The list of steps for the validation of the DFC stack simulation model is as follow:

- Conduct simulations on the DFC stack simulation model under the "ideal implementation", or the ideal environment where there is no randomness in timing of the events and no interferences from the host environment, and compare the message sequences in the simulation results against knowledge of the DFCstack obtained from specification documents and the test bench experiments.

- Create statistical models of the timing effects introduced by the host environment.

- Conduct simulations for all experiment scenarios in a simulated practical environment that closely resembles the host environment of the DFC stack (in the case of this thesis, the test bench), where the timing effects from the host environment are modeled. The simulations results are then compared against the nominal values and results from the test bench experiments.

The simulations under the ideal implementation is to validate the correctness of the logic of the simulation model, to ensure that the simulation model handles the transmit messages according to the specification documents, as discussed in Section 6.1. More specifically, the results is used to confirm whether the transmit messages and TX tasks are executed at expected times of arrival, whether the messages are enqueued and loaded into the TX buffers in the expected order, whether only eligible messages are loaded into the reserved slots and whether the delayed messages (transmit-pending messages that are not eligible for transmission in the current instance of their TX tasks) are assessed for queuing again in a later instance of their TX tasks.

As identified and analyzed from the test bench experiment data, the timing effects of the host environment have considerable impact on the behavior of the DFC stack. In order to create a practical environment that closely resembles the host environment of the actual system, statistical models of these timing effects are created based on the duration data from the experiments, and are used as input parameters to configure the simulation model. Then, simulations conducted with the combination of a logic-validated DFC stack simulation model and an simulated host environment will theoretically produce end results that can reflect the behaviors of the DFC running on the actual system, in other words, the simulation results should be identical or at

least very close to the test bench experiment results. Section 6.2 details the method used for creating the statistical models of the timing effects from the host environment, the simulations conducted in a simulated host environment incorporating these timing effects, and the validation of the simulation results, which proves the accuracy of the combination of the simulation model and the statistical models of the host environment against real-life data. The simulation results under the simulated practical environment will be validated against the test bench experiment results in terms of the transmit rates (TX rates) and output sequences of messages on the CAN bus (the message transmit rates will also be compared to their nominal values, which are defined off-line prior to the simulations and corresponding test bench experiments).

Simulations under both the ideal implementation and practical environment will consist of 2 different configurations of the DFC stack and 7 different sets of transmit messages for each configuration (pre-defined in the simulation script), which are identical to the test bench experiments. In other words, a total of 14 simulation scenarios will be simulated under both the ideal implementation and the practical environment. As explained in Section 4.1.3, for both configurations of the DFC stack, there are no reserved slots in Queue 0, and the transmit rate threshold for entering Queue 0 is set at 6.25 milliseconds so that only the 6.25ms TX task messages will be transmitted through Queue 0. While one of the configurations specifies 18 non-reserved slots in Queue 1 and that messages with transmit rates less than or equal to 75ms can enter the reserved slots, the other configuration specifies 19 non-reserved slots in Queue 1 and 50ms of entering transmit rate threshold.Figure 6.1 shows message sets including the message IDs, transmit rates and internal transmission priorities of the transmit messages. The message IDs are listed in the order of their internal transmission

priorities: the higher its position in the list, the higher the transmit priority.

Set 1		Set 2		Set 3		Set 4		Set 5		Set 6		Set 7	
ID (decimal)	Message Period (ms)	ID (decimal)	Message Period (ms)	ID (decimal)	Message Period (ms)	ID (decimal)	Message Period (ms)	ID (decimal)	Message Period (ms)	ID (decimal)	Message Period (ms)	ID (decimal)	Message Period (ms)
5	6.25	5	6.25	5	6.25	5	12.5	5	12.5	5	12.5	5	12.5
1	6.25	6	6.25	6	6.25	6	12.5	6	12.5	6	12.5	6	12.5
2	6.25	7	6.25	7	6.25	1	12.5	1	12.5	1	12.5	1	12.5
3	6.25	8	6.25	8	6.25	2	12.5	2	12.5	2	12.5	2	12.5
4	6.25	10	6.25	10	6.25	3	12.5	3	12.5	3	12.5	3	12.5
6	12.5	1	6.25	1	6.25	4	12.5	4	12.5	4	12.5	4	12.5
7	12.5	2	6.25	2	6.25	7	25	7	25	7	25	7	25
8	25	3	6.25	3	6.25	8	25	8	25	8	25	8	25
10	50	4	6.25	4	6.25	10	25	10	25	10	25	10	25
9	25	9	6.25	9	6.25	17	50	17	50	17	50	17	75
11	50	11	6.25	11	6.25	18	100	18	50	18	50	18	75
12	100	12	12.5	12	6.25	20	100	20	100	20	50	20	75
13	100	13	12.5	13	6.25	21	100	21	100	21	50	21	75
		14	12.5	14	6.25	9	25	22	100	22	50	22	75
		17	25	17	12.5	11	25	23	100	23	750	23	100
		18	50	15	12.5	12	50	9	25	24	750	24	100
		20	50	16	12.5	13	50	11	25	25	750	25	100
		21	100	18	25	14	50	12	25	26	750	26	100
		22	100	20	25	15	50	13	50	28	750	28	100
		23	100	21	50	16	50	14	50	9	25	9	25
		15	25	22	50	19	100	15	50	11	25	11	25
		16	25	23	50			16	50	12	25	12	25
		19	50	24	100			19	100	13	25	13	25
				25	100					14	25	14	25
				26	100					15	25	15	25
				19	25					16	25	16	25
										19	50	19	75
										27	750	27	100

Figure 6.1: The Message IDs and Nominal Transmit Rates of Transmit Messages in Each Message Set, Listed in the Order of Internal Transmission Priority.

In order to conveniently refer to different simulation scenarios, a code is assigned to each simulation scenario, which captures only the differences in the configurations of each scenario, more specifically, the calibrations in Queue 1 and the message set. Any simulation scenario is referred to as the "a-b-c simulation", where the number "a" stands for the number of non-reserved slots in Queue 1, the number "b" stands for the the transmit rate threshold for entering the reserved slots in Queue 1, and the number "c" is the message set number. For instance, for the simulation scenario that has 18 non-reserved slots in Queue 1, 75ms as transmit rate threshold for reserved

slots and message Set 3, it is called the "18-75-3 simulation".

6.1 Simulations in the Ideal Implementation

During simulations under the ideal implementation, the durations of all events in the DFC stack are determined based on deterministic models, and, in the case of this thesis, are set to constant values. More specifically, the durations of the transmission time of messages on the bus (message bus times) and the timing effects introduced by the host environment are specified as constant values to eliminate randomness in the system. For a realistic enqueue performance, the regular enqueue time of any message is set at 8 microseconds, the arithmetic mean of regular enqueue times computed from the test bench data. The irregular enqueue time is set at 0, which means the total duration of enqueue is not extended for message enqueues that are interrupted by the transmission-complete ISR. The inter-TX-task delays, t_i's, are set at 0, which means the execution of the TX tasks are not affected by RTOS scheduling. And the measurement jitter is also set at 0, which means that the RTOS causes no jitter in the captured transmission-complete ISR events and that the durations of measured message transmission times on the bus from the simulations are the same as the actual transmission times.

To simulate the bus time for each transmitted message, the DFC stack SimEvents model is connected to a SimEvents Single Server block, which simulates the CAN bus and stalls an entering message entity for a specific amount of time (the transmission time on the bus). All message transmitted in the ideal implementation have a DLC value of 8 (for consistent bus times for all messages and thus better readability in simulation results), and are always transmitted without bit stuffing. Similar to the

test bench experiments, all messages in the simulations will have standard identifiers, and the baudrate of the CAN bus is set at 500 kilobits per second. The total number of bits in a CAN data frame without bit stuffing is (47 + 8 * DLC), and thus the bus time of the data frame is then:

$$t_{bus} = \frac{47 + 8 * DLC}{baudrate}$$

At a baudrate of 500 kilobits per second, the bus time of a CAN frame is thus 222 microseconds. Hence the service time of the "CAN bus" Single Server block is set to 222 microseconds.

Since there is no randomness in simulations on the ideal implementation, only 1 simulation run is conducted for each simulation scenario. For each simulation scenario, simulation data is collected for a duration of 2 seconds of simulation time (in comparison, the trace captured in the test bench experiments only provides 1 second of data).

The ideal implementation is validated by comparing the model generated message sequences from all 14 simulation scenarios to the knowledge obtained from the specifications documents and hardware test bench traces. Comparison of the message sequence include the messages IDs of those messages that are placed into the transmit queues, the order in which they are placed and the order in which the messages are loaded into the TX buffers. The queuing assessment should control the queue depths according to the calibration values (Section 3.2.1), and the order in which messages are enqueued should follow the transmit priorities of those messages. The order of messages in the dequeue sequence should be consistent with the order of messages in the enqueue sequence due to the nature of the FIFO queues.

The comparison of the message sequences is performed on all captured events during a complete hyper-period of all the transmit messages, during which all transmit messages are expected to appear at least once on the CAN bus. The term "the start of a hyper-period" is defined as the point in time when all messages become pending for transmission, which only occurs when all TX tasks arrive at the same time. By this definition, the start of any hyper-period in a simulation, and in a test bench experiment, is the point in time where there is the largest number of transmit-pending messages, and thus the transmit queues have the most load. The starting point of the message sequence comparison is the start of the first hyper-period, after which the events captured in the trace are expected to repeat themselves once every duration of the hyper-period.

In message Sets 1, 2 and 4, the total number of transmit messages in each queue is no more than the maximum capacity of that queue (total number of slots in the queue plus the TX buffer). This ensures that even at start of a hyper-period, all transmit-pending messages will be eligible for transmission and thus no messages will be delayed. On the other hand, message Sets 3, 5, 6 and 7 have larger numbers of messages than the queue capacities, and it is quite likely that some of their messages will be delayed at the start of hyper-periods. Therefore, in order to demonstrate how the simulation model handles delayed messages, the thesis will provide detailed discussion on simulations with message Sets 3, 5 and 7 (message Set 6 is very similar to Set 7), more specifically, the 18-75-3, 19-50-5 and 19-50-7 simulations. Since the behavior of the reserved-slot mechanism with either DFC stack configuration can be well observed from a single simulation run, only one of the configuration is demonstrated in detail for the chosen message sets (for instance, only the 18-75-3 simulation

is discussed, but the 19-50-3 simulation is not). The behavior of the simulation model when there are no delayed messages is also demonstrated in the results of these simulations, from captured events that occur in other parts of the hyper-periods when only a subset of the transmit messages become pending for transmission. Message Sets 3, 5 and 7 have Queue 1 messages. Message Set 3 also has messages that are transmitted through Queue 0, which will demonstrates Queue 0 related behaviors.

6.1.1 The 18-75-3 Simulation

In the 18-75-3 simulation, the message set contains messages with periods of 6.25ms, 12.5ms, 25ms, 50ms and 100ms, which yields a hyper-period of 100ms. Figure 6.2 shows all the expected arrivals of TX tasks in a complete hyper-period in the 18-75-3 simulation, and Figure 6.3 shows the expected transmit-pending messages in each instance of the arrived TX tasks.

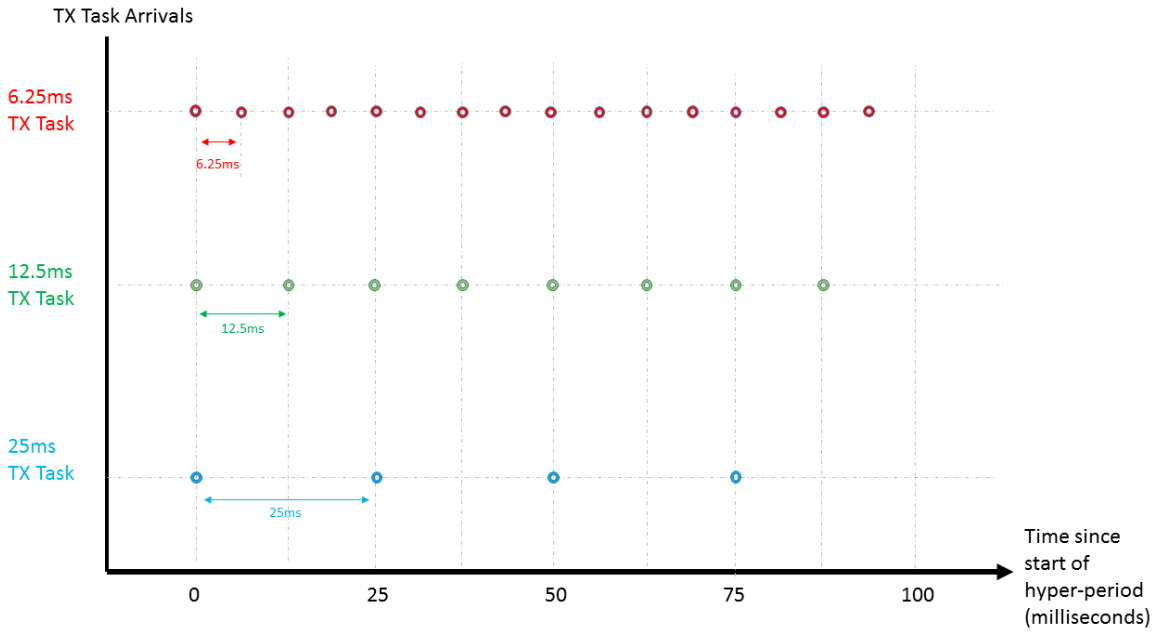

Figure 6.2: TX Task Arrivals During a Complete Hyper-Period in the 18-75-3 Simulation.

Time since start of hype-period (milliseconds)	Arrived TX Tasks	Arrived / Transmit-pending message IDs in TX task (In descending priority order from left to right)
0	6.25ms TX Task	All 14 messages (Message 5 through 14)
	12.5ms TX Task	All 3 messages (Message 17, 15, 16)
	25ms TX Task	All 9 messages (Message 18, 20, 21, 22, 23, 24, 25, 26, 19)
6.25	6.25ms TX Task	All 14 messages (Message 5 through 14)
12.5	6.25ms TX Task	All 14 messages (Message 5 through 14)
	12.5ms TX Task	All 3 messages (Message 17, 15, 16)
18.75	6.25ms TX Task	All 14 messages (Message 5 through 14)
25	6.25ms TX Task	All 14 messages (Message 5 through 14)
	12.5ms TX Task	All 3 messages (Message 17, 15, 16)
	25ms TX Task	Message 18, 20, 19
31.25	6.25ms TX Task	All 14 messages (Message 5 through 14)
37.5	6.25ms TX Task	All 14 messages (Message 5 through 14)
	12.5ms TX Task	All 3 messages (Message 17, 15, 16)
43.75	6.25ms TX Task	All 14 messages (Message 5 through 14)
50	6.25ms TX Task	All 14 messages (Message 5 through 14)
	12.5ms TX Task	All 3 messages (Message 17, 15, 16)
	25ms TX Task	Message 18, 20, 21, 22, 23, 19
56.25	6.25ms TX Task	All 14 messages (Message 5 through 14)
62.5	6.25ms TX Task	All 14 messages (Message 5 through 14)
	12.5ms TX Task	All 3 messages (Message 17, 15, 16)
68.75	6.25ms TX Task	All 14 messages (Message 5 through 14)
75	6.25ms TX Task	All 14 messages (Message 5 through 14)
	12.5ms TX Task	All 3 messages (Message 17, 15, 16)
	25ms TX Task	Message 18, 20, 19
81.25	6.25ms TX Task	All 14 messages (Message 5 through 14)
87.5	6.25ms TX Task	All 14 messages (Message 5 through 14)
	12.5ms TX Task	All 3 messages (Message 17, 15, 16)
93.75	6.25ms TX Task	All 14 messages (Message 5 through 14)

Figure 6.3: Expected Transmit-pending Messages in Each Instance of the TX Tasks.

Figure 6.4 and Figure 6.5 show a section of the model generated trace captured during the 18-75-3 simulation. The trace shows events that are related to the first instance of each TX task, which arrives at the start of the first hyper-period in the simulation run. Since the timers and transmit-pending flags of messages are reset at

the beginning of simulation (simulation time 0ms), the start of the first hyper-period in the 18-75-3 simulation is at simulation time 100ms.

Time (ms)	Q0 Enq ID	Q0 Deq ID / Enq Slot Num	Q1 Enq ID	Q1 Deq ID / Enq Slot Num	ISR Msg ID	Queue Depth	Num Msg Sent	Backward Time Diff
100		5				0	166	3.808
100.008	6	1				1	166	0.008
100.016	7	2				2	166	0.008
100.024	8	3				3	166	0.008
100.032	10	4				4	166	0.008
100.04	1	5				5	166	0.008
100.048	2	6				6	166	0.008
100.056	3	7				7	166	0.008
100.064	4	8				8	166	0.008
100.072	9	9				9	166	0.008
100.08	11	10				10	166	0.008
100.08				17		0	34	0
100.088			15	7		1	34	0.008
100.096			16	8		2	34	0.008
100.104			18	9		3	34	0.008
100.112			20	10		4	34	0.008
100.12			21	11		5	34	0.008
100.128			22	12		6	34	0.008
100.136			23	13		7	34	0.008
100.144			24	14		8	34	0.008
100.152			25	15		9	34	0.008
100.16			26	16		10	34	0.008
100.168			19	17		11	34	0.008
100.222					5	10	166	0.054
100.222		6				9	4	0
100.444					6	9	4	0.222
100.444		7				8	168	0
100.666					7	8	168	0.222
100.666		8				7	169	0
100.888					8	7	169	0.222
100.888		10				6	5	0
101.11					10	6	5	0.222
101.11		1				5	171	0
101.332					1	5	171	0.222
101.332		2				4	172	0
101.554					2	4	172	0.222
101.554		3				3	173	0
101.776					3	3	173	0.222
101.776		4				2	174	0

Figure 6.4: Model Generated Trace of the the 18-75-3 Simulation under the Ideal Implementation, from the Start of the First Hyper-period, Part 1

Time (ms)	Q0 Enq ID	Q0 Deq ID / Enq Slot Num	Q1 Enq ID	Q1 Deq ID / Enq Slot Num	ISR Msg ID	Queue Depth	Num Msg Sent	Backward Time Diff
101.776		4				2	174	0
101.998					4	2	174	0.222
101.998		9				1	175	0
102.22					9	1	175	0.222
102.22		11				0	176	0
102.442					11	0	176	0.222
102.664					17	11	34	0.222
102.664				15		10	35	0
102.886					15	10	35	0.222
102.886				16		9	36	0
103.108					16	9	36	0.222
103.108				18		8	37	0
103.33					18	8	37	0.222
103.33				20		7	38	0
103.552					20	7	38	0.222
103.552				21		6	39	0
103.774					21	6	39	0.222
103.774				22		5	40	0
103.996					22	5	40	0.222
103.996				23		4	41	0
104.218					23	4	41	0.222
104.218				24		3	42	0
104.44					24	3	42	0.222
104.44				25		2	43	0
104.662					25	2	43	0.222
104.662				26		1	44	0
104.884					26	1	44	0.222
104.884				19		0	45	0
105.106					19	0	45	0.222

Figure 6.5: Model Generated Trace of the the 18-75-3 Simulation under the Ideal Implementation, from the Start of the First Hyper-period, Part 2

Each line in the model generated trace represents an event. The first column is the absolute time in milliseconds since the start of the simulation. The blue-background columns show information of enqueue and dequeue events in Queue 0. In an enqueue event, the enqueued message ID is shown in the left of the 2 blue-background columns

and the target slot number in Queue 0 is shown in the right column; in a dequeue event, the ID of the message that is loaded into TX buffer 0 is shown on the right column while the left column is left blank (a TX buffer loading event of a message, which bypasses the queue and enters the TX buffer directly, shows up in the identical format in the trace). Similar to the blue-background columns, the green-background columns shows the enqueue and dequeue events in Queue 1. If a message ID is shown in the yellow-background column, it means that line is a transmission-complete ISR event, and the ID is of the message that has just been successfully transmitted. The last 3 right-most columns in the trace show the queue depth and total number of transmitted messages of the queue associated with the captured event, and the backward time difference between the current event and previous event in the trace.

At simulation time 100ms, the first of the 6.25ms messages, Message 5, is assessed for transmission. Since Queue 0 is empty and TX Buffer 0 is available, Message 5 bypasses Queue 0 and enter the TX buffer directly, as shown in the first line of the model generated trace in Figure 6.4. The second message, Message 6, is assessed right after the first TX buffer loading event and is loaded into Queue 0 after 8 microseconds, at simulation time 100.008ms. Message 6 is placed into Slot 1 (the slot number counts start from 1) of Queue 0 and the queue depth of Queue 0 is incremented to 1. This enqueue process continues for another 9 messages and stops after Message 11 is loaded into Queue 0 at simulation time 100.08ms. After this enqueue event, Message 11 has taken the last available slot in Queue 0 and Queue 0 has reached its full capacity. There are still 3 more 6.25ms TX task messages (Messages 12, 13 and 14) pending for transmission, however due a full queue they will not be able to enter the queue during this instance of the TX task. In fact, these 3 messages never get transmitted:

there is a total of 10 slots in Queue 0 and 14 messages pending for transmission in each instance of the 6.25ms TX task, the first message bypasses the queue and enters the TX buffer, the next 10 messages take up the entire queue space, and the last 3 messages are never eligible for transmission due to a full queue. The enqueue sequence so far for the Queue 0 messages are as expected the current combination of the DFC stack configuration and message set.

From simulation time 100.08ms, the queuing assessment starts for Queue 1 messages, which are managed by the 12.5ms and the 25ms TX tasks. The first 12.5ms TX task message, Message 17, bypasses Queue 1 and enters TX Buffer 1 since Queue 1 is empty and TX Buffer 1 is available. Then the rest of 12.5ms TX task messages and all 25ms TX task messages enters the queue in order of their transmit priorities. Since in message Set 3 the total number of transmit messages for Queue 1 is less than the total number of slots in Queue 1, all of these messages are eligible for transmission. There are a total of 12 transmit messages for Queue 1, the first message enters the TX buffer directly and the next 11 enter Queue 1, taking up Slots 7 through 17 in the case of this simulation (not always Slots 1 through 12 due to the nature of the ring buffer structure of the FIFO queues).

At simulation time 100.222ms, 222 microseconds after Message 5 is loaded into TX Buffer 0 for transmission on the bus, Message 5 has finished transmission and a transmission-complete ISR is generated. Message 6 is then loaded into TX Buffer 0 for transmission, and the queue depth decrements by 1. Between Message 6 in TX buffer 0 and Message 17 in TX Buffer 1, Message 6 wins arbitration on the CAN bus and starts transmission. After 222 microseconds, an ISR event is captured, indicating the completed transmission of Message 6. This continues until all Queue 0 messages

are transmitted on the bus, since the message IDs of Queue 0 messages always have smaller numerical values than the Queue 1 messages. Once the last Queue 0 message, Message 11, is successfully transmitted at simulation time 102.442ms, the Queue 1 messages are then transmitted on the bus, starting with Message 17, which has been previously loaded to TX Buffer 1.

In the rest of the trace captured during the 18-75-3 simulation, the TX tasks execute at their designated rates, the transmit messages arrive at expected instances of TX tasks, and the message sequences are in expected orders according to the DFC stack specifications.

6.1.2 The 19-50-5 Simulation

As message Set 5 contains transmit messages with periods of 12.5ms, 25ms, 50ms and 100ms, the duration of the hyper-period is 100ms. The expected arrivals of TX tasks within a hyper-period is the same as the 18-75-3 simulation (Figure 6.2). Since message Set 5 does not contain any 6.25ms messages, the 6.25ms TX task has zero execution time. The expected arrivals of the TX tasks and their transmit messages of a complete hyper-period in the 19-50-5 simulation are shown in Figure 6.6. For easier reference, a number is shown in the parentheses beside each instance of the TX tasks, separately numbering the instances of each TX task since the start of the hyper-period.

Time since start of hype-period (milliseconds)	Arrived TX Tasks (Num of instance)	Arrived / Transmit-pending message IDs in TX task (In descending priority order from left to right)
0	12.5ms TX Task (1)	All 6 messages (Message 5, 6, 1, 2, 3, 4)
	25ms TX Task (1)	All 17 messages (Message 7, 8, 10, 17, 18, 20, 21, 22, 23, 9, 11, 12, 13, 14, 15, 16, 19)
12.5	12.5ms TX Task (2)	All 6 messages (Message 5, 6, 1, 2, 3, 4)
25	12.5ms TX Task (3)	All 6 messages (Message 5, 6, 1, 2, 3, 4)
	25ms TX Task (2)	Message 7, 8, 10, 9, 11, 12
37.5	12.5ms TX Task (4)	All 6 messages (Message 5, 6, 1, 2, 3, 4)
50	12.5ms TX Task (5)	All 6 messages (Message 5, 6, 1, 2, 3, 4)
	25ms TX Task (3)	Message 7, 8, 10, 17, 18, 9, 11, 12, 13, 14, 15, 16
62.5	12.5ms TX Task (6)	All 6 messages (Message 5, 6, 1, 2, 3, 4)
75	12.5ms TX Task (7)	All 6 messages (Message 5, 6, 1, 2, 3, 4)
	25ms TX Task (4)	Message 7, 8, 10, 9, 11, 12
87.5	12.5ms TX Task (8)	All 6 messages (Message 5, 6, 1, 2, 3, 4)

Figure 6.6: Expected Transmit-pending Messages in Each Instance of the TX Tasks in the 19-50-5 Simulation.

Compared to the 18-75-3 simulation, both the 12.5ms and the 25ms TX tasks in the 19-50-5 simulation manage a larger number transmit messages. There are a total of 23 transmit-pending messages (6 messages from the 12.6ms TX task and 17 messages from the 25ms TX task) at the start of the first hyper-period in the 19-50-5 simulation. The total number of transmit-pending messages exceeds the maximum capacity of Queue1, and therefore some of these messages are expected to be delayed when they they become pending for transmission.

Figure 6.7 through Figure 6.10 show a section of the model generated trace captured during the 19-50-5 simulation. The section of the trace shows all events related to the TX task instances that arrive from simulation time 100ms to simulation time 125ms, during which all messages that have become transmit-pending at the start of the hyper-period are transmitted (some transmit-pending messages delayed to a later

instance of their TX tasks in which they become pending for transmission).

Time (ms)	Q0 Enq ID	Q0 Deq ID / Enq Slot Num	Q1 Enq ID	Q1 Deq ID / Enq Slot Num	ISR Msg ID	Queue Depth	Num Msg Sent	Backward Time Diff
100				5		0	67	11.168
100.008			6	20		1	67	0.008
100.016			1	1		2	67	0.008
100.024			2	2		3	67	0.008
100.032			3	3		4	67	0.008
100.04			4	4		5	67	0.008
100.048			7	5		6	67	0.008
100.056			8	6		7	67	0.008
100.064			10	7		8	67	0.008
100.072			17	8		9	67	0.008
100.08			18	9		10	67	0.008
100.088			20	10		11	67	0.008
100.096			21	11		12	67	0.008
100.104			22	12		13	67	0.008
100.112			23	13		14	67	0.008
100.12			9	14		15	67	0.008
100.128			11	15		16	67	0.008
100.136			12	16		17	67	0.008
100.144			13	17		18	67	0.008
100.152			14	18		19	67	0.008
100.16			15	19		20	67	0.008
100.222					5	20	67	0.062
100.222				6		19	68	0
100.444					6	19	68	0.222
100.444				1		18	69	0
100.666					1	18	69	0.222
100.666				2		17	70	0
100.888					2	17	70	0.222
100.888				3		16	71	0
101.11					3	16	71	0.222
101.11				4		15	72	0
101.332					4	15	72	0.222
101.332				7		14	73	0
101.554					7	14	73	0.222
101.554				8		13	74	0
101.776					8	13	74	0.222
101.776				10		12	75	0
101.998					10	12	75	0.222
101.998				17		11	76	0

Figure 6.7: Model Generated Trace of the 19-50-5 Simulation, Ideal Implementation, Part 1.

Time (ms)	Q0 Enq ID	Q0 Deq ID / Enq Slot Num	Q1 Enq ID	Q1 Deq ID / Enq Slot Num	ISR Msg ID	Queue Depth	Num Msg Sent	Backward Time Diff
101.998				17		11	76	0
102.22					17	11	76	0.222
102.22				18		10	77	0
102.442					18	10	77	0.222
102.442				20		9	78	0
102.664					20	9	78	0.222
102.664				21		8	79	0
102.886					21	8	79	0.222
102.886				22		7	80	0
103.108					22	7	80	0.222
103.108				23		6	81	0
103.33					23	6	81	0.222
103.33				9		5	82	0
103.552					9	5	82	0.222
103.552				11		4	83	0
103.774					11	4	83	0.222
103.774				12		3	84	0
103.996					12	3	84	0.222
103.996				13		2	85	0
104.218					13	2	85	0.222
104.218				14		1	86	0
104.44					14	1	86	0.222
104.44				15		0	87	0
104.662					15	0	87	0.222
112.5				5		0	88	7.838
112.508			6	20		1	88	0.008
112.516			1	1		2	88	0.008
112.524			2	2		3	88	0.008
112.532			3	3		4	88	0.008
112.54			4	4		5	88	0.008
112.722					5	5	88	0.182
112.722				6		4	89	0
112.944					6	4	89	0.222
112.944				1		3	90	0
113.166					1	3	90	0.222
113.166				2		2	91	0
113.388					2	2	91	0.222
113.388				3		1	92	0
113.61					3	1	92	0.222

Figure 6.8: Model Generated Trace of the 19-50-5 Simulation, Ideal Implementation, Part 2.

Time (ms)	Q0 Enq ID	Q0 Deq ID / Enq Slot Num	Q1 Enq ID	Q1 Deq ID / Enq Slot Num	ISR Msg ID	Queue Depth	Num Msg Sent	Backward Time Diff
113.388				3		1	92	0
113.61					3	1	92	0.222
113.61				4		0	93	0
113.832					4	0	93	0.222
125				5		0	94	11.168
125.008			6	5		1	94	0.008
125.016			1	6		2	94	0.008
125.024			2	7		3	94	0.008
125.032			3	8		4	94	0.008
125.04			4	9		5	94	0.008
125.048			7	10		6	94	0.008
125.056			8	11		7	94	0.008
125.064			10	12		8	94	0.008
125.072			9	13		9	94	0.008
125.08			11	14		10	94	0.008
125.088			12	15		11	94	0.008
125.096			16	16		12	94	0.008
125.104			19	17		13	94	0.008
125.222					5	13	94	0.118
125.222				6		12	95	0
125.444					6	12	95	0.222
125.444				1		11	96	0
125.666					1	11	96	0.222
125.666				2		10	97	0
125.888					2	10	97	0.222
125.888				3		9	98	0
126.11					3	9	98	0.222
126.11				4		8	99	0
126.332					4	8	99	0.222
126.332				7		7	100	0
126.554					7	7	100	0.222
126.554				8		6	101	0
126.776					8	6	101	0.222
126.776				10		5	102	0
126.998					10	5	102	0.222
126.998				9		4	103	0
127.22					9	4	103	0.222
127.22				11		3	104	0
127.442					11	3	104	0.222

Figure 6.9: Model Generated Trace of the 19-50-5 Simulation, Ideal Implementation, Part 3.

Time (ms)	Q0 Enq ID	Q0 Deq ID / Enq Slot Num	Q1 Enq ID	Q1 Deq ID / Enq Slot Num	ISR Msg ID	Queue Depth	Num Msg Sent	Backward Time Diff
127.442					11	3	104	0.222
127.442				12		2	105	0
127.664					12	2	105	0.222
127.664				16		1	106	0
127.886					16	1	106	0.222
127.886				19		0	107	0
128.108					19	0	107	0.222

Figure 6.10: Model Generated Trace of the 19-50-5 Simulation, Ideal Implementation, Part 4.

At simulation time 100ms, the first of the transmit-pending messages, Message 5 from the 12.5ms TX task, bypasses Queue 1 and enters the empty TX buffer 1. The enqueue sequence then starts with the rest of the transmit-pending messages: the 5 remaining messages from the 12.5ms TX task and then the 25ms TX task messages. Once the 12.5ms TX task messages have entered Queue 1, there are only 15 available slots left in the queue (20 slots in total, 5 occupied), which is not sufficient for the 17 transmit-pending messages from the 25ms TX task. As a result, at the end of the first instance of 25ms TX task, only 15 out of 17 messages have entered Queue 1 by simulation time 100.16ms. The 2 messages that are not eligible for transmission, Message 16 and 19, will be assessed for queuing again during the next instance of the 25ms TX task, which arrives at simulation time 125ms.One thing to note in the 19-50-5 simulation is the use of the reserved slot in Queue 1. By configuration, 19 out of 20 slots in Queue 1 are non-reserved slot, and thus, as Queue 1 is being filled up, the last available slot is a reserved slot. The last message loaded into Queue 1 in the enqueue sequence is Message 15 from the 25ms TX task. It has a nominal transmit-rate of 50ms, which is equal to the threshold for entering the reserved slots,

and thus it is eligible to enter the reserved slot (the criterion for entering a reserved slot is to be below or equal to the threshold). The order of message enqueuing is identical to internal transmit priority of these messages.

Once Message 5 is successfully transmitted on the bus, indicated by the transmission-complete ISR event at simulation time 100.222ms, the dequeue sequence of Queue 1 starts. The messages in Queue 1, Message 6 through Message 15, are dequeued consecutively in the order of their enqueue events. The last enqueued message that becomes pending for transmission at the beginning of the hyper-period, Message 15, completes its transmission at simulation time 104.662ms.

At simulation time 112.5ms, the second instance of the 12.5ms TX task arrives and starts transmitting its messages. As Queue 1 has been emptied and the TX buffer is available, Message 5 again bypasses Queue 1 and enters the TX buffer directly. Then the rest of the 12.5ms TX task messages enter Queue 1, and are transmitted in the expected order.

At simulation time 125ms, the third instance of the 12.5ms TX task and the second instance of the 25ms TX task arrive at the same time. All the 12.5ms TX task messages become pending for transmission as usual, while only 6 of the 25ms TX task messages become pending for transmission, more specifically, the 6 messages that have a common transmit rate of 25ms. Aside from these 6 messages, Message 16 and 19 are still pending for transmission since the previous instance of the 25ms TX task. As a result, a total of 14 messages are transmit-pending at simulation time 125ms (6 from the 12.5ms TX tasks and 8 from the 25ms TX task). The first message, Message 5, enters TX buffer 1 directly as Queue 1 is empty and the TX buffer is available. The rest of the transmit-pending messages are all able to enter

Queue 1 as all 20 slots in Queue 1 are available. After the enqueue sequence from simulation time 125ms to 125.104ms, the transmit-pending messages are loaded into Queue 1 in the order of their transmit priorities. From simulation time 125.222, the dequeue sequence starts. The enqueued messages are transmitted on the bus in the order of their enqueue events.

Since Message 16 and 19 are not eligible for transmission at the start of the hyper-period and are delayed till the second instance of the 25ms TX task, their timers and transmit-pending flags are not reset until the second instance of the 25ms task, which is executed at simulation time 125ms. Consequently, Message 16, which has a nominal transmit rate of 50, will become transmit-pending again at simulation time 175ms. Hypothetically, if Message 16 was successfully transmitted in the first instance of the 25ms TX task, it would become transmit-pending at simulation time 150ms. The delayed transmission of Message 16 has created a phase difference of 25ms since the start of the first hyper-period. Instead of predicted arrivals shown in Figure 6.6, Message 16 will be transmit-pending at simulation time 175ms, 225ms, 275, *etc.*. The same effect is created on arrivals of Message 19, which will be transmit-pending at simulation time 225ms, 325ms, 425ms, *etc.* instead of 200ms, 300ms, 400ms, *etc.*

In the rest of the trace, the TX tasks execute at their designated rates, the transmit messages arrive at expected instances of the TX tasks, and the message sequences are in expected orders according to the DFC stack specifications.

6.1.3 The 19-50-7 Simulation

The message Set 7 contains transmit messages with periods of 12.5ms, 25ms, 75ms and 100ms, and thus the duration of the hyper-period is 300ms. Similar to the 19-50-5

simulation, the 19-50-7 simulation has a number of Queue 1 messages that exceeds the maximum capacity of slots in Queue 1. Consequently, it is also very likely in the 19-50-7 simulation that some of the transmit-pending messages at the start of a hyper-period will be delayed due to a full Queue 1. Based on the simulation results of the 19-50-7 simulation, it appears that the number of Queue 1 messages in the 19-50-7 simulation is actually so large that there are messages being delayed at points in time other than the start of a hyper-period. Detailed explanation will be discussed below.

First, Figures 6.11 and 6.12 show the model generated trace at the start of the first hyper-period in the 19-50-7 simulation (at simulation time 300ms), which shows the message sequences that are related to only the 1^{st} instances of the 12.5ms and 25ms TX tasks.

Time (ms)	Q0 Enq ID	Q0 Deq ID / Enq Slot Num	Q1 Enq ID	Q1 Deq ID / Enq Slot Num	ISR Msg ID	Queue Depth	Num Msg Sent	Backward Time Diff
300				5		0	278	11.168
300.008			6	15		1	278	0.008
300.016			1	16		2	278	0.008
300.024			2	17		3	278	0.008
300.032			3	18		4	278	0.008
300.04			4	19		5	278	0.008
300.048			7	20		6	278	0.008
300.056			8	1		7	278	0.008
300.064			10	2		8	278	0.008
300.072			17	3		9	278	0.008
300.08			18	4		10	278	0.008
300.088			20	5		11	278	0.008
300.096			21	6		12	278	0.008
300.104			22	7		13	278	0.008
300.112			23	8		14	278	0.008
300.12			24	9		15	278	0.008
300.128			25	10		16	278	0.008
300.136			26	11		17	278	0.008
300.144			28	12		18	278	0.008
300.152			9	13		19	278	0.008
300.16			11	14		20	278	0.008
300.222					5	20	278	0.062
300.222				6		19	279	0
300.444					6	19	279	0.222
300.444				1		18	280	0
300.666					1	18	280	0.222
300.666				2		17	281	0
300.888					2	17	281	0.222
300.888				3		16	282	0
301.11					3	16	282	0.222
301.11				4		15	283	0
301.332					4	15	283	0.222
301.332				7		14	284	0
301.554					7	14	284	0.222
301.554				8		13	285	0
301.776					8	13	285	0.222
301.776				10		12	286	0
301.998					10	12	286	0.222
301.998				17		11	287	0

Figure 6.11: Model Generated Trace of Ideal Implementation, the 19-50-7 Simulation, from the Start of the First Hyper-period, Part 1.

Time (ms)	Q0 Enq ID	Q0 Deq ID / Enq Slot Num	Q1 Enq ID	Q1 Deq ID / Enq Slot Num	ISR Msg ID	Queue Depth	Num Msg Sent	Backward Time Diff
301.998				17		11	287	0
302.22					17	11	287	0.222
302.22				18		10	288	0
302.442					18	10	288	0.222
302.442				20		9	289	0
302.664					20	9	289	0.222
302.664				21		8	290	0
302.886					21	8	290	0.222
302.886				22		7	291	0
303.108					22	7	291	0.222
303.108				23		6	292	0
303.33					23	6	292	0.222
303.33				24		5	293	0
303.552					24	5	293	0.222
303.552				25		4	294	0
303.774					25	4	294	0.222
303.774				26		3	295	0
303.996					26	3	295	0.222
303.996				28		2	296	0
304.218					28	2	296	0.222
304.218				9		1	297	0
304.44					9	1	297	0.222
304.44				11		0	298	0
304.662					11	0	298	0.222

Figure 6.12: Model Generated Trace of Ideal Implementation, the 19-50-7 Simulation, from the Start of the First Hyper-period, Part 2.

As expected, the first transmit-pending message, Message 5, bypasses Queue 1 and enters the TX buffer directly, and Messages 6 through 11 enter Queue 1 in the order of their internal transmit priorities. Once Message 11 has entered the queue, all slots in Queue 1 have been occupied (including the reserved slot, which is occupied by Message 11). Consequently, the rest of the 25ms TX task messages, Messages 12 through 27, which are expected to be pending for transmission, are not loaded in

the first instance of the 25ms TX task. Since Messages 12, 13, 14, 15 and 16 have transmit-rates for 25ms, they will naturally appear in the 2^{nd} instance of the 25ms TX task no matter being delayed or not, while Messages 19 and 27 should appear as an addition to the rest of the messages since they are expected to be pending for transmission.

Figure 6.13 shows the enqueue events that are performed during the 2^{nd} instance of the 25ms TX task (arrived at simulation time 325ms) in the model generated trace.

Time (ms)	Q0 Enq ID	Q0 Deq ID / Enq Slot Num	Q1 Enq ID	Q1 Deq ID / Enq Slot Num	ISR Msg ID	Queue Depth	Num Msg Sent	Backward Time Diff
325				5		0	305	11.168
325.008			6	20		1	305	0.008
325.016			1	1		2	305	0.008
325.024			2	2		3	305	0.008
325.032			3	3		4	305	0.008
325.04			4	4		5	305	0.008
325.048			7	5		6	305	0.008
325.056			8	6		7	305	0.008
325.064			10	7		8	305	0.008
325.072			9	8		9	305	0.008
325.08			11	9		10	305	0.008
325.088			12	10		11	305	0.008
325.096			13	11		12	305	0.008
325.104			14	12		13	305	0.008
325.112			15	13		14	305	0.008
325.12			16	14		15	305	0.008
325.128			19	15		16	305	0.008

Figure 6.13: Model Generated Trace for the 2^{nd} Instance of the 25ms TX Task.

After the enqueue events for the 12.5ms TX task messages, the transmit-pending 25ms TX task messages are then loaded into Queue 1. Message 19, which was expected to become transmit-pending from the previous instance of the 25ms TX task, is the last enqueued message in this instance of the 25ms TX task. The other message

that is also expected to be transmit-pending, Message 27, is still not yet enqueued even though Queue 1 still has 4 available slots. In fact, Message 27 is loaded into Queue 1 during the 3^{rd} instance of the 25ms TX task since the start of the first hyper-period, which arrives at simulation time 350ms, and the enqueue sequence in that instance (together with the 12.5ms TX task instance that arrives at the same time) is shown in Figure 6.14 below.

Time (ms)	Q0 Enq ID	Q0 Deq ID / Enq Slot Num	Q1 Enq ID	Q1 Deq ID / Enq Slot Num	ISR Msg ID	Queue Depth	Num Msg Sent	Backward Time Diff
350				5		0	328	11.168
350.008			6	1		1	328	0.008
350.016			1	2		2	328	0.008
350.024			2	3		3	328	0.008
350.032			3	4		4	328	0.008
350.04			4	5		5	328	0.008
350.048			7	6		6	328	0.008
350.056			8	7		7	328	0.008
350.064			10	8		8	328	0.008
350.072			9	9		9	328	0.008
350.08			11	10		10	328	0.008
350.088			12	11		11	328	0.008
350.096			13	12		12	328	0.008
350.104			14	13		13	328	0.008
350.112			15	14		14	328	0.008
350.12			16	15		15	328	0.008
350.128			27	16		16	328	0.008

Figure 6.14: Model Generated Trace for the 3^{rd} Instance of the 25ms TX Task.

This outcome of Message 27 is because the fact that its earlier arrivals before the start of the first hyper-period have been delayed, and thus its arrivals after the delay have a phase difference from predicted arrivals assuming no delays. Message 27 is not enqueued during the 2^{nd} instance of the 25ms TX task while there are still available slots in Queue 1, because it has not become transmit-pending at simulation time

325ms. Thus it is loaded into Queue 1 during the 3^{rd} instance of 25ms TX task as it becomes transmit-pending at simulation time 350ms. To demonstrate the cause of this outcome, sections of the model generated trace before the start of the first period will be discussed below. With a nominal transmit rate of 100ms, Message 27 first becomes pending for transmission at simulation time 100ms. Figure 6.15 shows the enqueue sequence in Queue 1 during TX tasks that arrive at simulation time 100ms.

Time (ms)	Q0 Enq ID	Q0 Deq ID / Enq Slot Num	Q1 Enq ID	Q1 Deq ID / Enq Slot Num	ISR Msg ID	Queue Depth	Num Msg Sent	Backward Time Diff
100				5		0	78	11.168
100.008			6	11		1	78	0.008
100.016			1	12		2	78	0.008
100.024			2	13		3	78	0.008
100.032			3	14		4	78	0.008
100.04			4	15		5	78	0.008
100.048			7	16		6	78	0.008
100.056			8	17		7	78	0.008
100.064			10	18		8	78	0.008
100.072			23	19		9	78	0.008
100.08			24	20		10	78	0.008
100.088			25	1		11	78	0.008
100.096			26	2		12	78	0.008
100.104			28	3		13	78	0.008
100.112			9	4		14	78	0.008
100.12			11	5		15	78	0.008
100.128			12	6		16	78	0.008
100.136			13	7		17	78	0.008
100.144			14	8		18	78	0.008
100.152			15	9		19	78	0.008
100.16			16	10		20	78	0.008

Figure 6.15: The Enqueue Sequence of TX Tasks Arrived at Simulation Time 100ms.

By simulation time 100.16ms, the 20 slots in Queue 1 has been taken by Messages 6 through 16. Message 27 is not eligible for transmission since the queue is full, and it is therefore delayed for assessment until the next instance of the 25ms TX task,

which arrives at simulation time 125ms. Figure 6.16 shows the enqueue sequence in Queue 1 since simulation time 125ms.

Time (ms)	Q0 Enq ID	Q0 Deq ID / Enq Slot Num	Q1 Enq ID	Q1 Deq ID / Enq Slot Num	ISR Msg ID	Queue Depth	Num Msg Sent	Backward Time Diff
125				5		0	105	11.168
125.008			6	16		1	105	0.008
125.016			1	17		2	105	0.008
125.024			2	18		3	105	0.008
125.032			3	19		4	105	0.008
125.04			4	20		5	105	0.008
125.048			7	1		6	105	0.008
125.056			8	2		7	105	0.008
125.064			10	3		8	105	0.008
125.072			9	4		9	105	0.008
125.08			11	5		10	105	0.008
125.088			12	6		11	105	0.008
125.096			13	7		12	105	0.008
125.104			14	8		13	105	0.008
125.112			15	9		14	105	0.008
125.12			16	10		15	105	0.008
125.128			19	11		16	105	0.008
125.136			27	12		17	105	0.008

Figure 6.16: The Actual Arrival of the 1^{st} Instance of Message 27.

At simulation time 125.136ms, Message 27 is successfully loaded into Queue 1, and thus its timer and transmit-pending flag are reset. Consequently, Message 27 will become transmit-pending again in 100ms, or, at simulation time 225ms. The enqueue sequence during the TX tasks that arrive at simulation time 225ms is shown in Figure 6.17.

Time (ms)	Q0 Enq ID	Q0 Deq ID / Enq Slot Num	Q1 Enq ID	Q1 Deq ID / Enq Slot Num	ISR Msg ID	Queue Depth	Num Msg Sent	Backward Time Diff
225				5		0	205	11.168
225.008			6	8		1	205	0.008
225.016			1	9		2	205	0.008
225.024			2	10		3	205	0.008
225.032			3	11		4	205	0.008
225.04			4	12		5	205	0.008
225.048			7	13		6	205	0.008
225.056			8	14		7	205	0.008
225.064			10	15		8	205	0.008
225.072			17	16		9	205	0.008
225.08			18	17		10	205	0.008
225.088			20	18		11	205	0.008
225.096			21	19		12	205	0.008
225.104			22	20		13	205	0.008
225.112			9	1		14	205	0.008
225.12			11	2		15	205	0.008
225.128			12	3		16	205	0.008
225.136			13	4		17	205	0.008
225.144			14	5		18	205	0.008
225.152			15	6		19	205	0.008
225.16			16	7		20	205	0.008

Figure 6.17: The Enqueue Sequence of TX Tasks Arrived at Simulation Time 225ms.

During the instance of the 25ms TX task that arrives at simulation time 225ms, messages with transmit rates of 75ms become pending for transmission, and they are assessed for queuing before Message 27 due to higher internal transmit priorities. Consequently, Queue 1 if fully filled before Message 27 is assessed, and Message 27 is delayed again until the next instance of the 25ms TX task, which arrives at simulation time 250ms. And as expected, Message 27 is successfully enqueued during the 25ms TX task arrived at simulation time 250ms, as shown in the enqueue sequence in Figure 6.18 below. As a result, after the enqueue, Message 27 is then expected to become transmit-pending at simulation time 350ms, but not at 300ms.

Time (ms)	Q0 Enq ID	Q0 Deq ID / Enq Slot Num	Q1 Enq ID	Q1 Deq ID / Enq Slot Num	ISR Msg ID	Queue Depth	Num Msg Sent	Backward Time Diff
250				5		0	232	11.168
250.008			6	13		1	232	0.008
250.016			1	14		2	232	0.008
250.024			2	15		3	232	0.008
250.032			3	16		4	232	0.008
250.04			4	17		5	232	0.008
250.048			7	18		6	232	0.008
250.056			8	19		7	232	0.008
250.064			10	20		8	232	0.008
250.072			9	1		9	232	0.008
250.08			11	2		10	232	0.008
250.088			12	3		11	232	0.008
250.096			13	4		12	232	0.008
250.104			14	5		13	232	0.008
250.112			15	6		14	232	0.008
250.12			16	7		15	232	0.008
250.128			19	8		16	232	0.008
250.136			27	9		17	232	0.008

Figure 6.18: The Actual Arrival of the 2^{nd} Instance of Message 27.

The generated trace from the 19-50-7 simulation shows that the DFC stack simulation model is able to produce expected message sequences when having a relatively large quantity of transmit messages under the ideal implementation.

The simulations under the ideal implementation has demonstrated expected message sequences in all 14 simulation scenarios, and thus validate the correctness of the logic of the DFC stack according to the specification documents.

6.2 Simulations in Practical Environment

This section first details the method for creating the statistical models of the timing effects from the host environment, and then discusses the simulation results and the

validation of the simulation model under the simulated practical environment.

The simulation model is capable of simulating 4 types of timing effects from the host environment based on their statistical models that are specific to the target host environment. The timing effects are the inter-TX-task delays (t_i's), the measurement jitter, the regular and the irregular enqueue times of transmit messages. From the visual observation of the test bench experiment data, the durations of these timing effects seem to be random, and it is more reasonable to simulate them by using statistical models than deterministic models such as constant values. To create a statistical model for each of the timing effects, it is necessary to first determine the existence of correlation among the data points, and then, once confirmed that there is no correlation (i.e. the durations of a timing effect are independent from each other), a suitable probability distribution is chosen to model the durations. The correlation among the data points is analyzed based on the autocorrelation plot created through the MATLAB function *xcorr*. As for choosing the most suitable probability distribution, 3 candidate probability distributions are created (fitted on the duration data through the MATLAB function *fitdist*) and compared to the distribution of the duration data. The candidate probability distributions include 2 parametric distributions (the normal distribution and the gamma distribution) and 1 non-parametric distribution (the kernel distribution). If the distribution of the duration data of a timing effect appears to be in an uncommon shape that is visually different from the parametric distributions, the kernel distribution is used to model that timing effect. Otherwise, the chi-squared test is performed to determine which parametric distribution between the normal and gamma distributions best resembles the duration data of the timing effect.

Due to the existence of randomness in using statistical models (random number generation, *etc.*), it is desirable to gather as much simulation data as possible, in order to gain more confidence that the captured data can closely reflect the behavior of the simulation model. However, due to limited access to the research facility at the industrial partner, it was not possible to gather sufficient data from the industrial test bench to allow a more rigorous analysis of the statistical nature of the data in order to be able to precisely determine how many simulation runs would be needed. As a result, for simulation data gathering, it was decided to conduct simulations in the MATLAB environment for a specified duration of time that requires a reasonable amount of effort to complete. More specifically, simulations conducted under the practical environment takes around 5 hours, during which 20 simulation runs have been completed for each simulation scenario and each simulation run captures data for 2 seconds of simulation time. In contrast to the test bench experiment data, which provides only a single 1-second experiment for each scenario, the simulations provide as many as 40 times the events captured during test bench experiments.

Different from the simulations under the ideal environment, where the DFC stack SimEvents model is connected to a SimEvents Single Server block that simulates constant bus times for all transmit message, an instance of the CAN Bus blocked implemented in the Vector CAN stack library is deployed instead of the single server block in order to simulate a more realistic CAN bus, in which the bus times of messages are computed based on their DLCs (Whinton (2016)). For simulations under the simulated practical environment, two aspects of the simulation results are validated against the test bench experiment data: (i) the transmit rates of the transmit message (also compared to nominal values) and (ii) the output message sequence on the bus.

6.2.1 Statistical Models of the Regular and Irregular Enqueue Times

The durations of the message enqueue times are profiled from the traces of all 14 experiment scenarios and are combined into 3 sets of data: the Queue 0 regular enqueue times, Queue 1 regular enqueue times and the Queue 1 irregular enqueue times.

As observed from the test bench experiment data, the irregular enqueues have only occurred in Queue 1. It appears that the transmission time of a single message on the bus is longer than filling Queue 0 with messages from the empty state to full capacity (enqueuing 10 messages into a total of 10 slots in Queue 0), and consequently no message enqueue has been interrupted by a transmission-complete ISR and irregular enqueues have not occurred in Queue 0.

To determine the correlation among the durations of the message enqueue times, the normalized autocorrelation plots of the Queue 0 regular enqueue times, Queue 1 regular enqueue times and the Queue 1 irregular enqueue times are shown in Figures 6.19, 6.20 and 6.21, respectively.

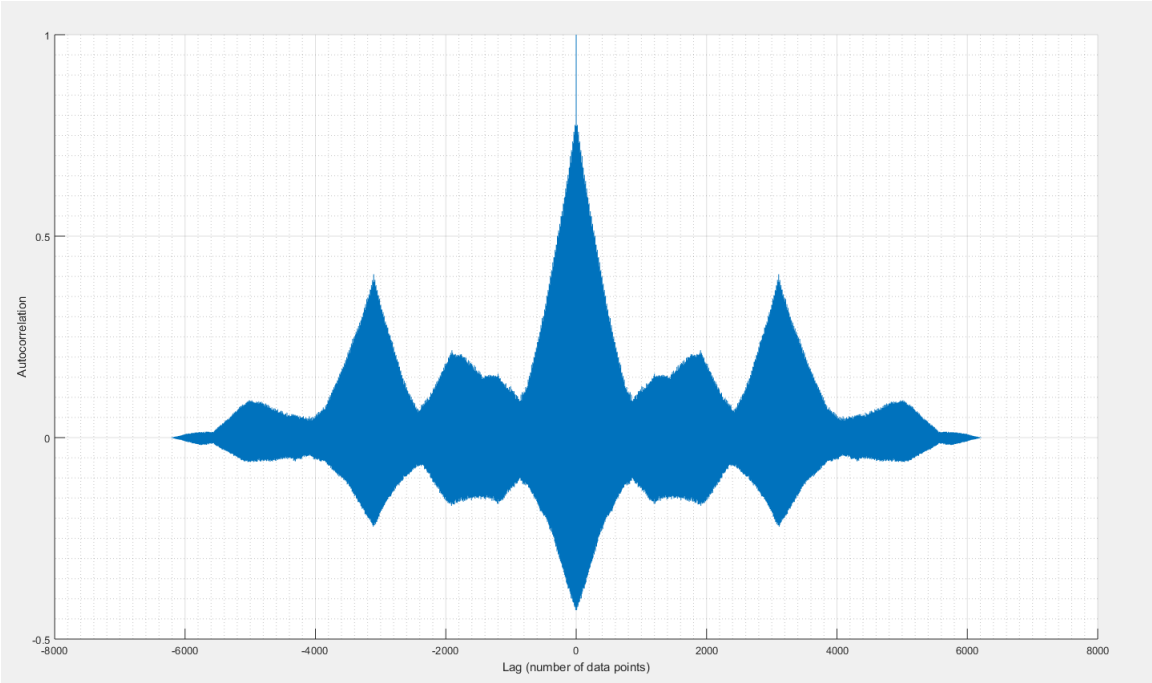

Figure 6.19: Autocorrelation of Normalized Regular Enqueue Times in Queue 0.

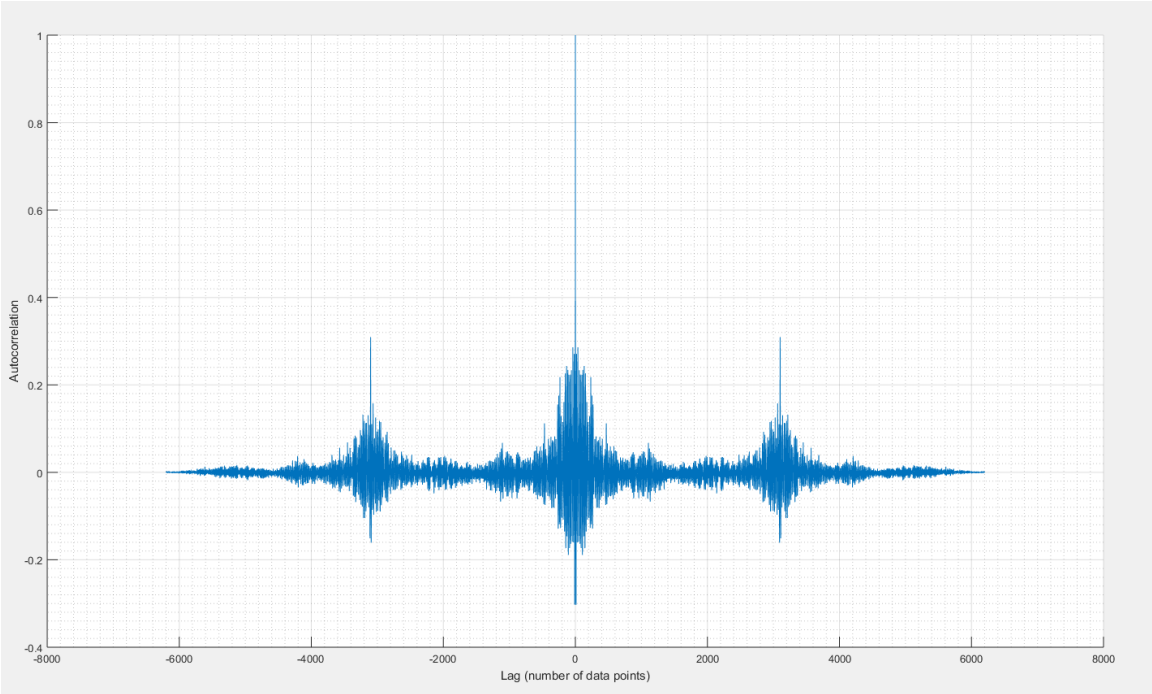

Figure 6.20: Autocorrelation of Normalized Regular Enqueue Times in Queue 1.

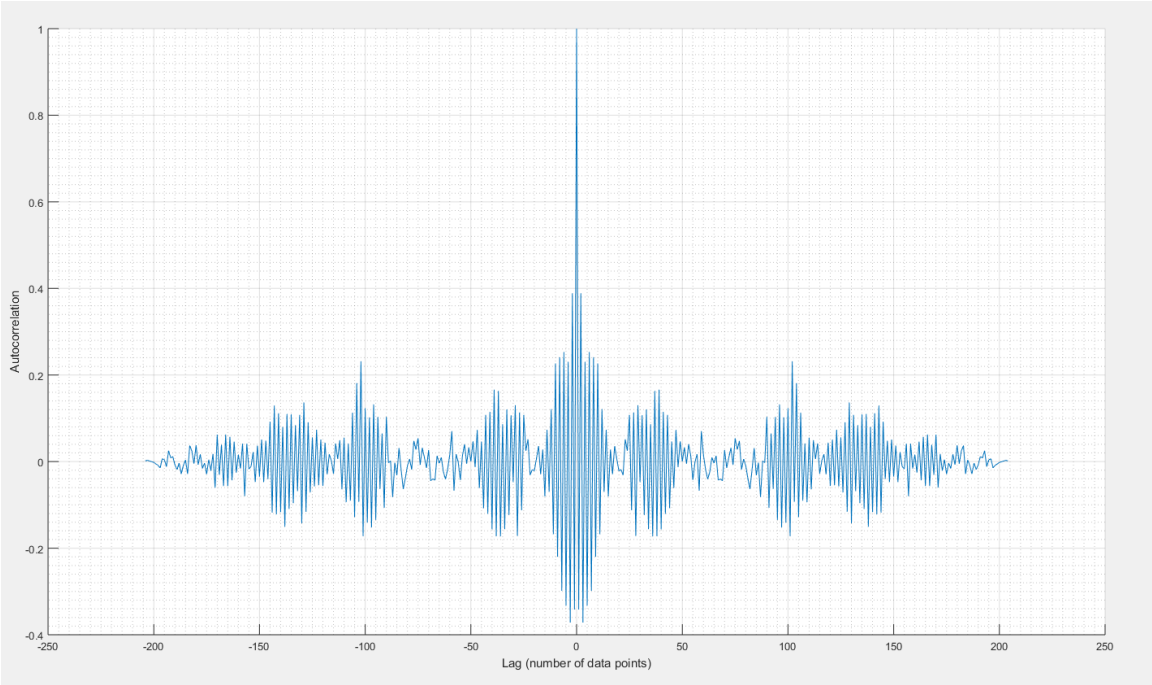

Figure 6.21: Autocorrelation of Normalized Irregular Enqueue Times in Queue 0.

The y-axes in the figures show the normalized autocorrelation, and the x-axes show the lag in number of data points. In the plots, all 3 sets of data have an autocorrelation value of "1" when the lag is "0", and a low autocorrelation value (well below "1") anywhere else, which indicates that the data sets are independent and there is no correlation among the data points in each set.

The next step is to find the probability distribution that best resembles the actual data. To visually compare between the candidate probability distributions, Figure 6.22 shows the normal distribution, the gamma distribution and the kernel distribution along with the histogram of the actual duration data of the regular enqueue times in Queue 0.

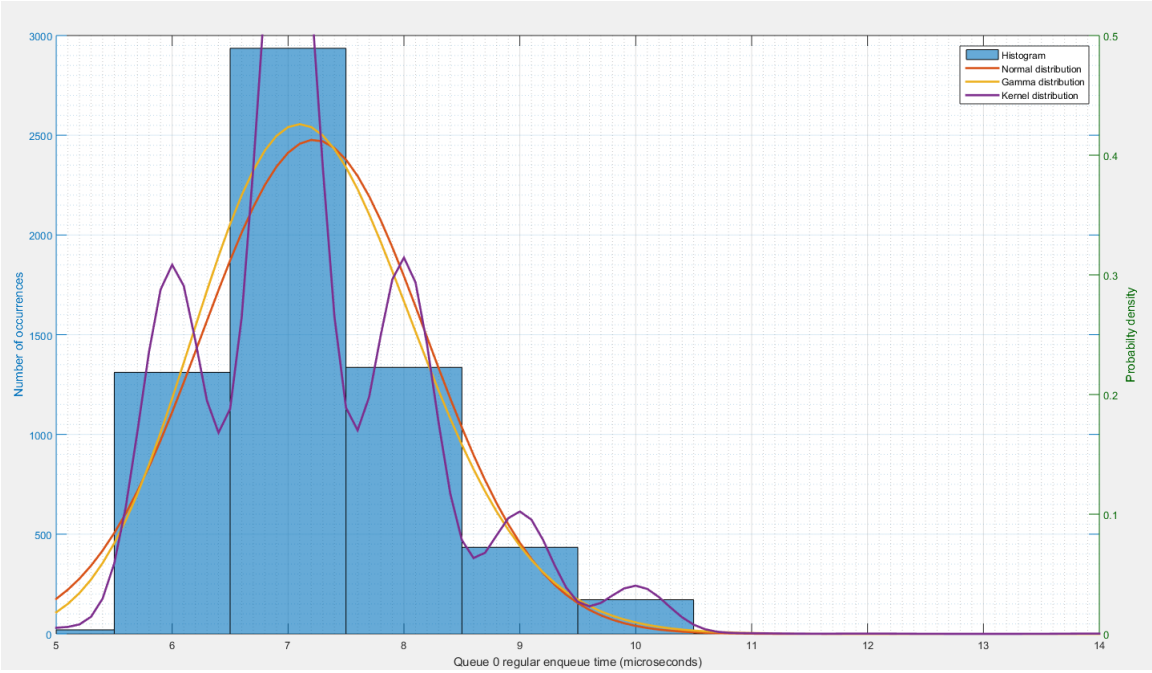

Figure 6.22: Candidate Probability Density Functions and the Histogram of Regular Enqueue Times in Queue 0.

The x-axis displays the enqueue times in microseconds. The histogram of enqueue times is plotted in blue vertical bars and number of occurrences indicated by the bars is shown on the left y-axis. The red, yellow and purple lines are the probability density functions of the normal, gamma and kernel distributions, respectively, that are fitted on the actual duration data of the regular enqueue times in Queue 0. As shown in the figure, the histogram appears to be close to the shape of a bell curve, which is more similar to the parametric distributions (normal or the gamma distribution) than the kernel distribution.Therefore the durations of the Queue 0 regular enqueue times are to be modeled by either the normal or gamma distribution, based on the result of a chi-squared test between each probability distribution and the actual data. From the chi-squared tests for the normal and the gamma distribution, while the

null hypothesis is rejected for both, the probability value (p value) for the gamma distribution is smaller than that of the normal distribution, which indicates that the gamma distribution better resembles the actual data. As a result, the gamma distribution is used to model the durations of the Queue 0 regular enqueue times.

Figures 6.23 and 6.24 show the fitted probability density functions versus histogram plots of the regular and irregular enqueue times in Queue 1, respectively.

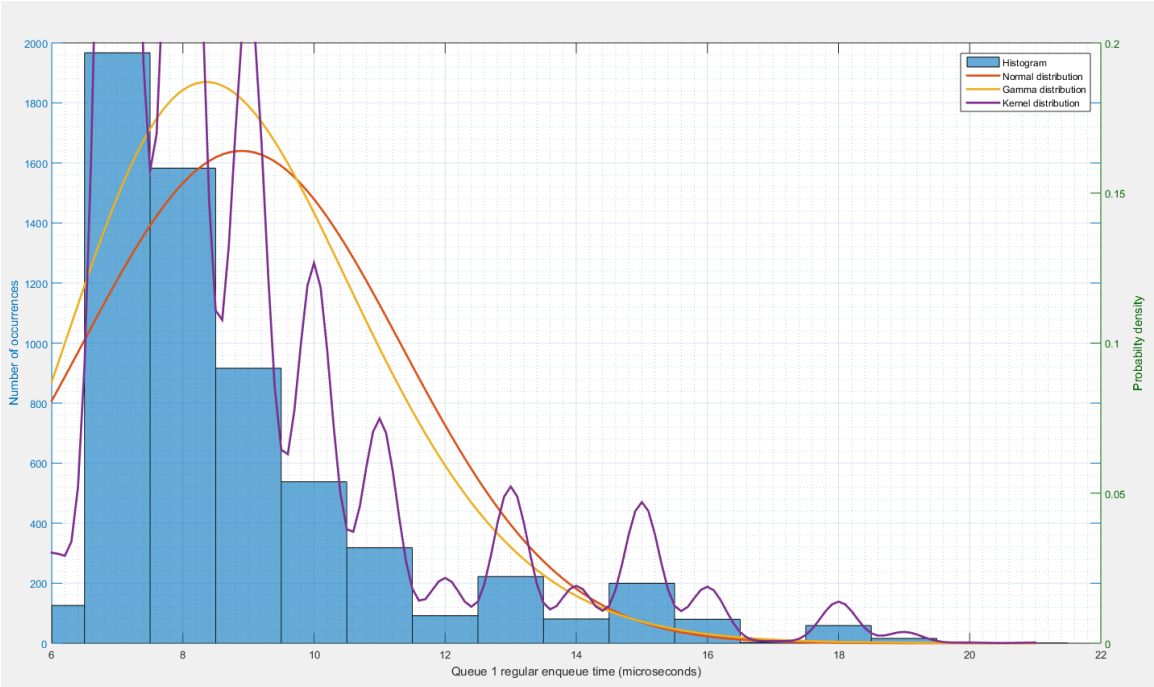

Figure 6.23: Candidate Probability Density Functions versus Histogram of Regular Enqueue Times in Queue 1.

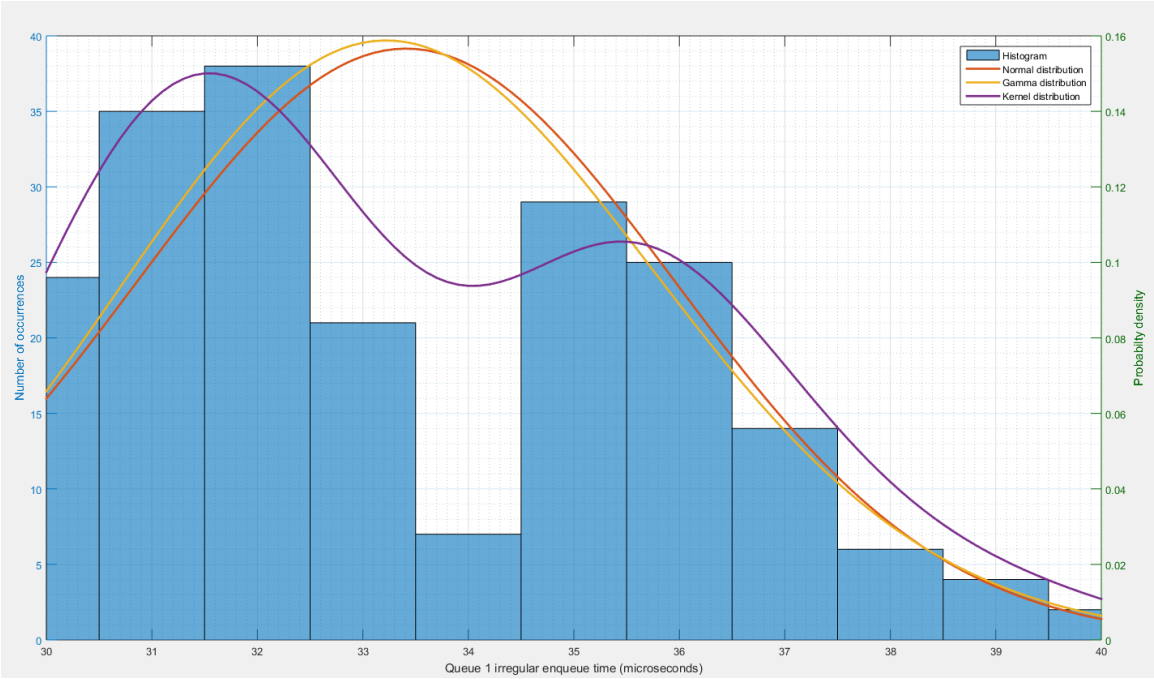

Figure 6.24: Candidate Probability Density Functions versus Histogram of Irregular Enqueue Times in Queue 0.

The gamma distribution is used for modeling the regular enqueue times in Queue 1, for similar reason when deciding the probability distribution for modeling the Queue 0 regular enqueue times. However, for the irregular enqueue times in Queue 1, the histogram of the durations has a double-hump shape, which appears different than the parametric distribution candidates. As a results the kernel distribution function is used for modeling the durations of the irregular enqueue times in Queue 1.

6.2.2 Statistical Model of the Inter-TX-Task Delays

Figures 6.25 and 6.26 shows the autocorrelation of the profiled durations of the $t_{i,1}$'s (inter-TX-task delays between the 6.25ms and 12.5ms TX tasks) and $t_{i,2}$'s (inter-TX-task delays between the 12.5ms and 25ms TX tasks), respectively.

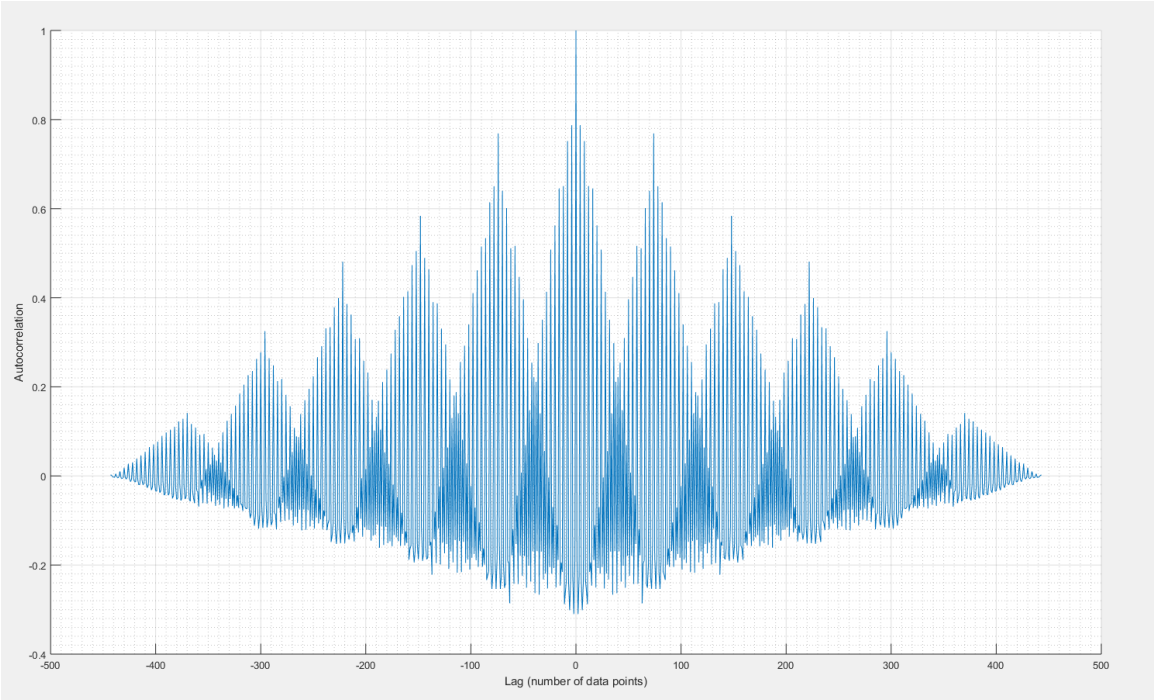

Figure 6.25: Autocorrelation of Normalized $t_{i,1}$.

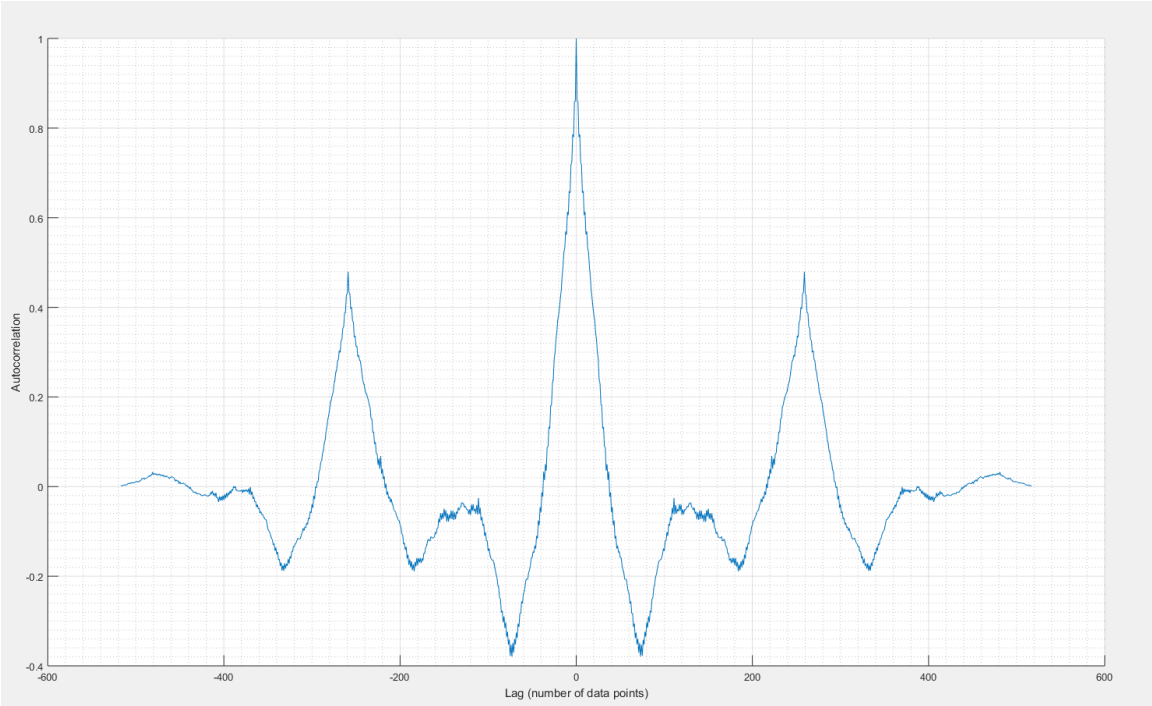

Figure 6.26: Autocorrelation of Normalized $t_{i,2}$.

From the autocorrelation plots, we can conclude that the inter-TX-task delays are independent and there is no correlations among the profiled $t_{i,1}$ or $t_{i,2}$ values, since for both sets of data the autocorrelation has a value "1" only when the lag is "0".

Figures 6.27 and 6.28 below show the candidate probability density functions versus histogram plots of $t_{i,1}$ and $t_{i,2}$, respectively.

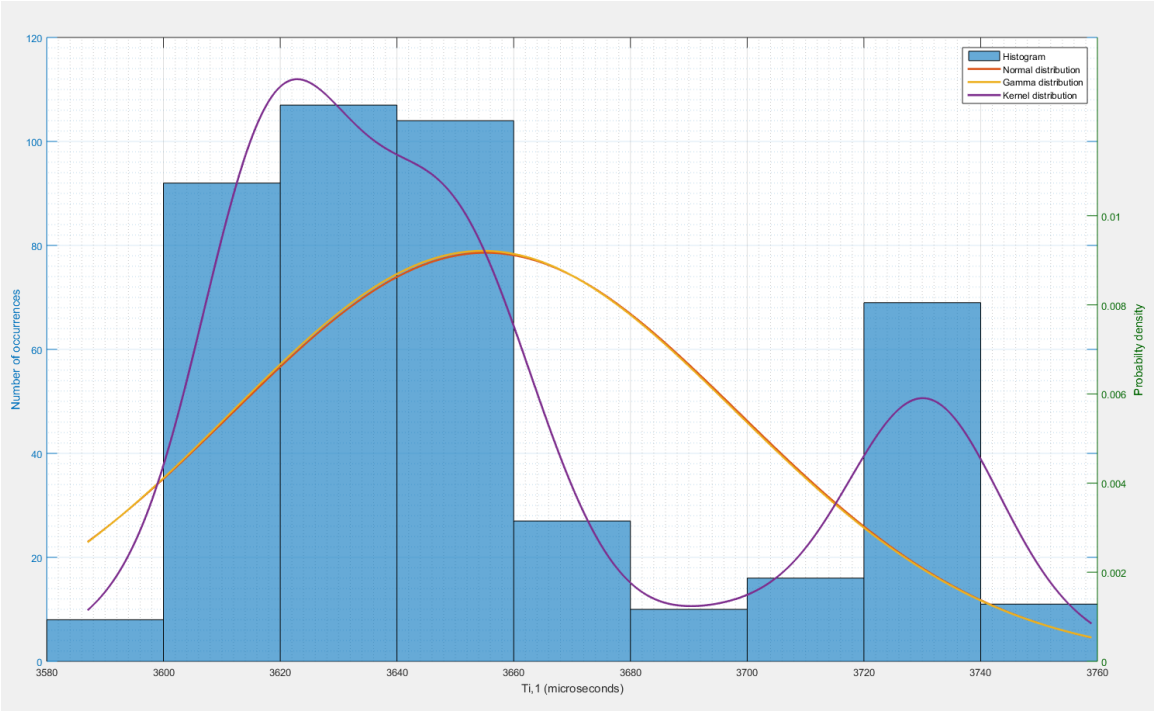

Figure 6.27: Candidate Probability Density Functions versus Histogram of $t_{i,1}$.

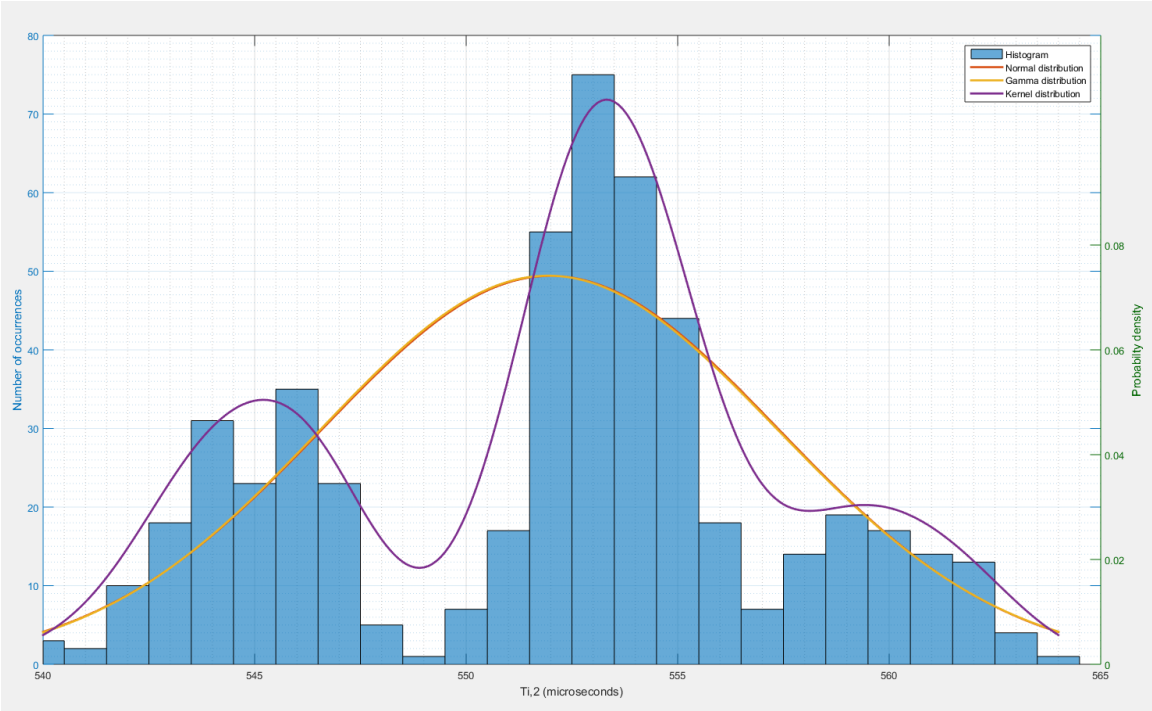

Figure 6.28: Candidate Probability Density Functions versus Histogram of $t_{i,2}$.

For both $t_{i,1}$ and $t_{i,2}$, there multiple humps in the shape of their histograms. Consequently, the normal and gamma distributions, which projects relatively simpler shapes, are not as suitable as the kernel distribution. Therefore, the kernel distribution is used to model both $t_{i,1}$ and $t_{i,2}$ from the host environment.

6.2.3 Statistical Model of the Measurement Jitter

The measurement jitter is applied by the host environment to each individual successful message transmission. The amount of profiled measurement jitter durations is considerably larger than other types of timing effects from the host environment. Before testing the autocorrelation on the large amount of data, it is helpful to first split the entire set of measurement jitter data into smaller sets based on a characteristic

that divides the transmitted messages, and check if the smaller sets have different data patterns that are potentially caused by that characteristic. Thus, the duration data is first split into several separated sets based on the transmit rates of the transmit messages. In other words, the measurement jitter durations in each set are profiled from transmit messages of the same nominal transmit rate. The existing transmit rates in the 7 message sets are 6.25ms, 12.5ms, 25ms, 50ms, 75ms, 100ms and 750ms. Thus the measurement jitter data are first split into 7 sets. After plotting the histogram of the jitter durations in each set, the shapes of these histograms appear to be similar, and distribution of the durations in all data sets appears to congregate within a very similar range. Therefore, it is concluded that the measurement jitter caused by the host environment is not dependent on the nominal transmit rate of the messages, and it is safe to analyzes the autocorrelation among the the collective durations of the measurement jitter across messages with different nominal transmit rates. Figure 6.29 below shows the autocorrelation plot of the combined measurement jitter durations, and the plot concludes that the measurement jitter durations from the test bench experiments are independent.

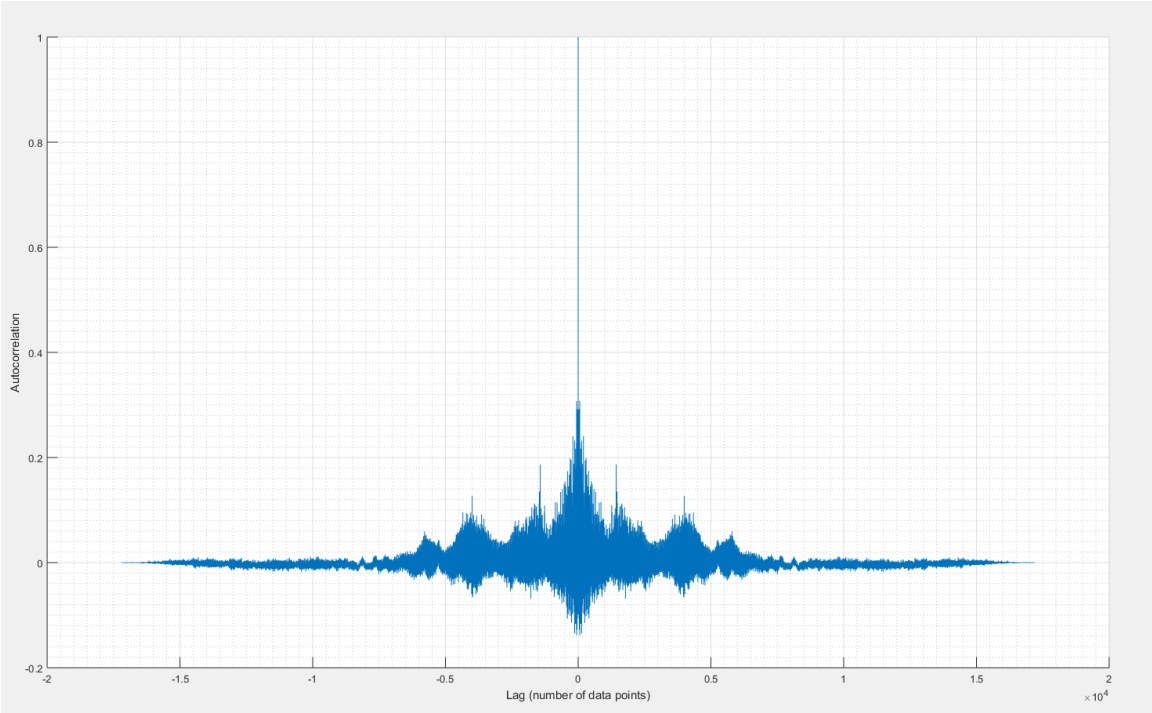

Figure 6.29: Autocorrelation of Normalized Measurement Jitter.

The probability density functions of the candidate probability distributions versus histogram of the measurement jitter is plotted in Figure 6.30 below. Due to the double-hump shaped histogram, the kernel distribution is a better fit compared to the normal and gamma distributions. Thus the kernel distribution is used for modeling the measurement jitter in the simulated practical environment.

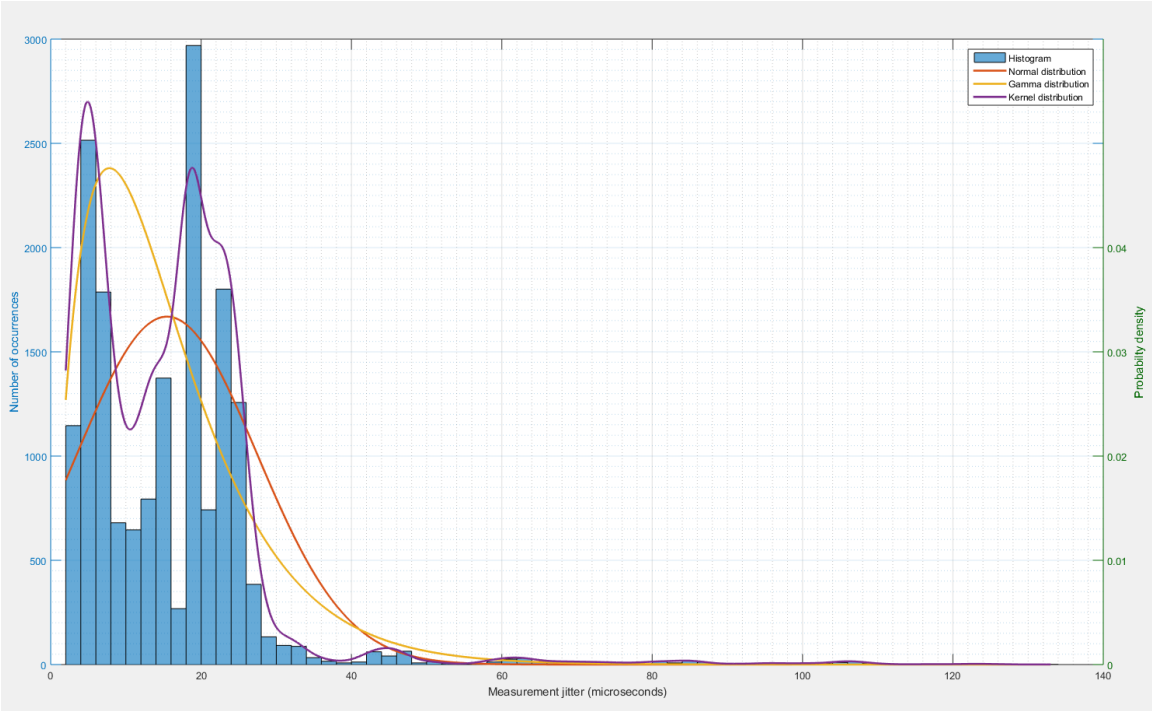

Figure 6.30: Candidate Probability Density Functions versus Histogram of Measurement Jitter.

6.2.4 Transmit Rates of Messages

For validation of message transmit rates under the simulated practical environment, the transmit rates of the messages in the simulations ("simulated transmit rates") will be compared to the nominal transmit rates and the profiled transmit rates from the corresponding test bench experiment trace ("test bench transmit rates"). The simulated and the test bench transmit rates are compared between each simulation scenario and its corresponding test bench experiment scenario which has the same DFC stack configuration. The definition of "accurate transmit rate in a simulation" compared to the nominal or the test bench transmit rate requires that the 95% confidence interval of the simulated transmit rate of a message ID contains the nominal

or the test bench transmit rate within its bounds, respectively, where the the 95% confidence interval is computed by applying the MATLAB function *paramci* ("confidence interval for probability distribution parameters") on the normal distribution fitted on the data of the simulated transmit rate.

The validation procedure is shown in Figure 6.31. For each simulation scenario, 20 simulation runs are conducted, and thus there is a total of (20 * 14 = 280) simulation runs in a complete validation procedure of the message transmit rates. From each single simulation run, the numbers of messages IDs, whose simulated transmit rate 95% confidence intervals contains its nominal or test bench transmit rates within the bounds, called the "number of containment" of the nominal or the test bench transmit rates, are gathered and are added together to compute an overall percentage, the "percentage of containment", which is calculated by dividing the total numbers of containment by the total number of occurrences of all message IDs, and the percentage of containment is the quantitative result of the validation. The data collection process will be explained further below by different stage in the validation procedure.

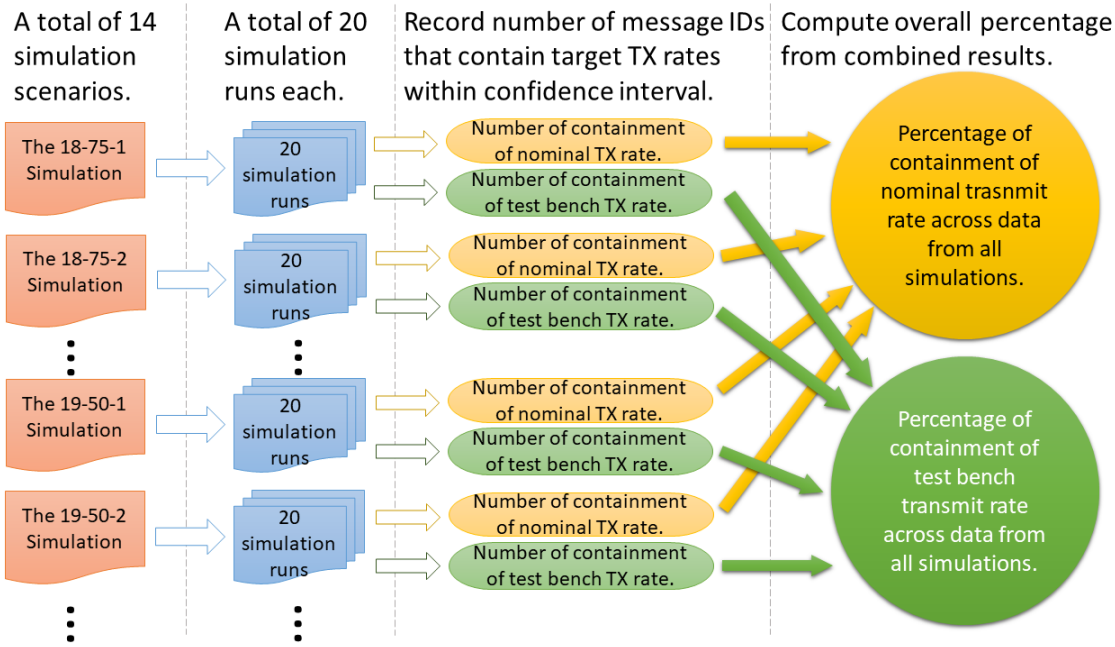

Figure 6.31: The Data Collection Process for Message Transmit Rates Validation.

The first step is the calculation of the message transmit rates of the simulations and the test bench experiments (the nominal transmit rates are pre-defined for the experiment scenarios and require no calculation). The transmit rate of a message is calculated by taking the arithmetic mean of the inter-arrival times of that message on the bus, and the inter-arrival times are computed by taking the backward time difference of every 2 consecutive transmission-complete ISR events associated with the same message ID. This is done for both the simulated and the test bench transmit rates. And then, the 95% confidence interval of the simulated transmit rate of message is computed from a normal distribution fitted on the inter-arrival times of that message message during a simulation (normal distribution is fitted on the inter-arrival times through the MATLAB function *fitdist*, and the confidence interval is calculated through the MATLAB function *paramci*). For both the simulation model

and the test bench setup, the transmit rate calculation is based on the timestamps of the transmission-complete ISR events in the traces, which records the completion times of successful transmissions of each message. In contrast, the dequene events are not used for the inter-arrival time calculation because the completion time of the TX buffer loading event of a message does not always reflect the time that a message appears on the bus (such as when a message is loaded into the TX buffer while the CAN bus is not idle, *etc*).

For each simulation run, the statistics related to both the simulated and the corresponding test bench message transmit rates are computed and saved in a spreadsheet. Figure 6.32 shows the transmit rate statistics of a single simulation run of the 18-75-3 simulation scenario.

Message ID (base 10)	Model TX rate, arithmetic mean (ms)	Model TX rate, minimum (ms)	Model TX rate, maximum (ms)	Sufficient data points?	Model TX rate 95% percent confidence interval, lower bound (ms)	Model TX rate 95% percent confidence interval, upper bound (ms)	Nominal TX rate (ms)	Contained within confidence interval?	Test bench experiment TX rate (ms)	Contained within confidence interval?
5	6.250123	6.164241	6.339787	Yes	6.2483785	6.2518671	6.25	Yes	6.2509456	Yes
6	6.250071	6.157937	6.339318	Yes	6.2476656	6.2524764	6.25	Yes	6.250932	Yes
7	6.250023	6.152153	6.340415	Yes	6.2470891	6.2529569	6.25	Yes	6.2509252	Yes
8	6.250037	6.129647	6.346641	Yes	6.2466452	6.253429	6.25	Yes	6.250932	Yes
10	6.250007	6.123596	6.398239	Yes	6.2463346	6.2536799	6.25	Yes	6.250932	Yes
1	6.250015	6.109434	6.388846	Yes	6.2456945	6.2543356	6.25	Yes	6.2509456	Yes
2	6.250007	6.108514	6.378685	Yes	6.24536	6.2546539	6.25	Yes	6.2509388	Yes
3	6.250091	6.120851	6.382579	Yes	6.2453533	6.2548285	6.25	Yes	6.2509456	Yes
4	6.250073	6.108209	6.38192	Yes	6.2449002	6.2552454	6.25	Yes	6.2509456	Yes
9	6.250043	6.11469	6.362592	Yes	6.2447682	6.2553185	6.25	Yes	6.2512945	Yes
11	6.250062	6.109587	6.379583	Yes	6.2445297	6.2555936	6.25	Yes	6.2513014	Yes
12	0	0	0	No	N/A	N/A	6.25	N/A	0	N/A
13	0	0	0	No	N/A	N/A	6.25	N/A	0	N/A
14	0	0	0	No	N/A	N/A	6.25	N/A	0	N/A
17	12.50074	12.3452	12.65449	Yes	12.489576	12.511901	12.5	Yes	12.502959	Yes
15	12.50087	12.31741	12.65584	Yes	12.488973	12.512772	12.5	Yes	12.502959	Yes
16	12.50085	12.27109	12.6862	Yes	12.488707	12.512989	12.5	Yes	12.502945	Yes
18	25.00026	24.84482	25.16077	Yes	24.983001	25.017526	25	Yes	25.004806	Yes
20	25.00118	24.85191	25.15443	Yes	24.98356	25.018801	25	Yes	25.00475	Yes
21	49.9997	49.80604	50.15141	Yes	49.970588	50.028804	50	Yes	50.0095	Yes
22	50.00011	49.79777	50.17751	Yes	49.969527	50.030693	50	Yes	50.009556	Yes
23	49.99989	49.77504	50.17397	Yes	49.969089	50.030688	50	Yes	50.014556	Yes
24	99.99951	99.91075	100.0851	Yes	99.973185	100.02584	100	Yes	100.02989	No
25	99.99869	99.86539	100.1352	Yes	99.965154	100.03222	100	Yes	100.02789	Yes
26	99.99734	99.85693	100.1268	Yes	99.960755	100.03393	100	Yes	100.02589	Yes
19	24.98576	23.73677	26.27281	Yes	24.780253	25.191261	25	Yes	25.006472	Yes

Figure 6.32: Message Transmit Rate Validation Statistics from a Single Simulation Run of the 18-75-3 Simulation Scenario.

The first column in Figure 6.32 lists the message IDs of the transmit message set in the 18-75-3 simulation scenario. The message IDs are listed in descending order of the internal transmit priorities. Columns 2, 3 and 4, which have uncolored background, list the arithmetic mean, minimum and maximum of the inter-arrival times of messages in the simulation run. (The maximum inter-arrival times can be used for worst-case scenario analysis of message response times, where the maximum inter-arrival time of messages are compared to their deadlines to see if any deadlines

are missed.) The fifth column in the spreadsheet indicates if the message ID of each row has appeared at least 10 times on the bus. This is necessary since the probability distribution fitting function ($fitdist$) in MATLAB requires at least 10 data samples, and if the number of occurrences of a message is too low the statistics will be considerably less meaningful. Therefore, if a message ID does not have sufficient data (i.e. appear on the bus for less than 10 times), no comparison with the confidence interval will be made and there will be a "NO" in the fifth column, and thus this message ID is not eligible to be used towards the overall result, the percentage of containment. The columns in green background show the lower and upper bounds of the 95% confidence interval of the simulated transmit rate of each transmit message. The columns in yellow background show the nominal transmit rate of each message and whether the simulated transmit rate confidence interval of that message contains the nominal transmit rate within its bounds. And similarly, the blue-background columns show the test bench experiment transmit rates and containment of the 95% confidence interval of the simulated transmit rates.

In this particular spreadsheet, it appears that Message IDs 12, 13 and 14 do not have sufficient data to be compared with the nominal and the test bench transmit rates. This is because, as mentioned in Section 6.1.1, the maximum capacity of Queue 0 during a single execution of the 6.25ms TX task is 11 and there are 14 message in the 6.25ms TX task in the 18-75-3 simulation scenario, and consequently the 3 messages with the lowest priorities never get transmitted. This outcome is also agreed by the test bench trace of the 18-75-3 experiment, as no events related to these message IDs are captured in the test bench trace (indicated by a "0" in the first blue-background column, where it shows the test bench experiment transmit rates). And, as expected,

the transmit rate validation result from any simulations of the 19-50-3 simulation scenario, which has the same transmit message set, also shows the same outcome, as shown in Figure 6.33.

Message ID (base 10)	Model TX rate, arithmetic mean (ms)	Model TX rate, minimum (ms)	Model TX rate, maximum (ms)	Sufficient data points?	Model TX rate 95% percent confidence interval, lower bound (ms)	Model TX rate 95% percent confidence interval, upper bound (ms)	Nominal TX rate (ms)	Contained within confidence interval?	Test bench experiment TX rate (ms)	Contained within confidence interval?
5	6.249975	6.046207	6.46933	Yes	6.2454563	6.2544944	6.25	Yes	6.2510476	Yes
6	6.25002	6.047035	6.462037	Yes	6.2450375	6.2550021	6.25	Yes	6.2510272	Yes
7	6.249967	6.042453	6.456598	Yes	6.2446371	6.2552977	6.25	Yes	6.2510272	Yes
8	6.249937	6.031991	6.450238	Yes	6.2442947	6.255579	6.25	Yes	6.2510272	Yes
10	6.249904	6.033943	6.454727	Yes	6.2441005	6.2557084	6.25	Yes	6.2510272	Yes
1	6.249875	6.020569	6.468061	Yes	6.2436157	6.2561342	6.25	Yes	6.2510408	Yes
2	6.249887	6.019554	6.467797	Yes	6.243559	6.2562142	6.25	Yes	6.2510408	Yes
3	6.249901	6.020113	6.487939	Yes	6.2431709	6.2566319	6.25	Yes	6.2510408	Yes
4	6.249968	6.005484	6.489047	Yes	6.2428689	6.2570664	6.25	Yes	6.2510408	Yes
9	6.24999	5.996477	6.508736	Yes	6.2426838	6.2572953	6.25	Yes	6.2512671	Yes
11	6.249996	6.013055	6.527667	Yes	6.242562	6.2574293	6.25	Yes	6.251274	Yes
12	0	0	0	No	N/A	N/A	6.25	N/A	0	N/A
13	0	0	0	No	N/A	N/A	6.25	N/A	0	N/A
14	0	0	0	No	N/A	N/A	6.25	N/A	0	N/A
17	12.50032	12.3423	12.6901	Yes	12.488079	12.512562	12.5	Yes	12.502959	Yes
15	12.50025	12.34533	12.70019	Yes	12.487906	12.512604	12.5	Yes	12.502973	Yes
16	12.50036	12.31541	12.68211	Yes	12.487776	12.512936	12.5	Yes	12.502973	Yes
18	25.00033	24.81867	25.25552	Yes	24.981919	25.018742	25	Yes	25.004833	Yes
20	25.00029	24.82811	25.25288	Yes	24.981262	25.019326	25	Yes	25.00475	Yes
21	49.99975	49.8081	50.16944	Yes	49.973066	50.026429	50	Yes	50.009556	Yes
22	49.99914	49.79932	50.18454	Yes	49.972756	50.025522	50	Yes	50.009556	Yes
23	50.0002	49.79301	50.18127	Yes	49.972415	50.027988	50	Yes	50.0145	Yes
24	99.99687	99.89719	100.1439	Yes	99.966827	100.02691	100	Yes	100.03067	No
25	99.99693	99.88094	100.1465	Yes	99.965948	100.0279	100	Yes	100.02944	No
26	99.99727	99.89771	100.1408	Yes	99.966762	100.02778	100	Yes	100.02767	Yes
19	24.94604	20.69669	29.29341	Yes	24.279068	25.613022	25	Yes	25.006917	Yes

Figure 6.33: Transmit Message Transmit Rate Validation Statistics from a Simulation Run of the 19-50-3 Simulation Scenario.

As shown in both Figure 6.32 and Figure 6.33, in the columns that indicate the numbers of containment (the right of the yellow-background and blue-background columns), most of the message IDs have a simulated transmit rate confidence interval

that contains the nominal and test bench values within the bounds. However, for Message 24 in Figure 6.32 and Messages 24 and 25 in Figure 6.33, their test bench transmit rates lie outside of the 95% confidence intervals of there simulated transmit rates. Because timing effects from the host environment are modeled with pseudo-random number generation, it is normal that the simulated transmit rates and their confidence intervals fluctuate, and thus occasionally the confidence intervals do not contain the nominal and/or test bench transmit rates within the bounds. And to demonstrate other possible outcomes, Figures 6.34 and 6.35 show statistics of a different simulation run for the 18-75-3 and 19-50-3 simulation scenarios, respectively, in which, for *all* messages IDs, the simulated transmit rate confidence intervals contain both the nominal and test bench transmit rates within their bounds.

Message ID (base 10)	Model TX rate, arithmetic mean (ms)	Model TX rate, minimum (ms)	Model TX rate, maximum (ms)	Sufficient data points?	Model TX rate 95% percent confidence interval, lower bound (ms)	Model TX rate 95% percent confidence interval, upper bound (ms)	Nominal TX rate (ms)	Contained within confidence interval?	Test bench experiment TX rate (ms)	Contained within confidence interval?
5	6.25003	6.15879	6.35032	Yes	6.2479952	6.2520638	6.25	Yes	6.2509456	Yes
6	6.250095	6.15278	6.35305	Yes	6.2473845	6.2528063	6.25	Yes	6.250932	Yes
7	6.250169	6.15114	6.35175	Yes	6.2469557	6.2533816	6.25	Yes	6.2509252	Yes
8	6.250096	6.12398	6.36993	Yes	6.2463502	6.2538416	6.25	Yes	6.250932	Yes
10	6.250164	6.12437	6.36233	Yes	6.2458997	6.2544282	6.25	Yes	6.250932	Yes
1	6.250208	6.12107	6.39934	Yes	6.2453694	6.2550463	6.25	Yes	6.2509456	Yes
2	6.250143	6.1244	6.38303	Yes	6.2451115	6.2551742	6.25	Yes	6.2509388	Yes
3	6.25018	6.11316	6.4349	Yes	6.2446793	6.255681	6.25	Yes	6.2509456	Yes
4	6.25005	6.03477	6.43637	Yes	6.244184	6.2559155	6.25	Yes	6.2509456	Yes
9	6.250066	6.03834	6.45804	Yes	6.2440213	6.2561106	6.25	Yes	6.2512945	Yes
11	6.250044	6.04608	6.44537	Yes	6.2437034	6.2563837	6.25	Yes	6.2513014	Yes
12	0	0	0	No	N/A	N/A	6.25	N/A	0	N/A
13	0	0	0	No	N/A	N/A	6.25	N/A	0	N/A
14	0	0	0	No	N/A	N/A	6.25	N/A	0	N/A
17	12.50002	12.3507	12.6466	Yes	12.489181	12.510854	12.5	Yes	12.502959	Yes
15	12.50016	12.3461	12.6723	Yes	12.488908	12.511404	12.5	Yes	12.502959	Yes
16	12.5002	12.2962	12.658	Yes	12.488718	12.511674	12.5	Yes	12.502945	Yes
18	25.00097	24.7855	25.1475	Yes	24.983881	25.018066	25	Yes	25.004806	Yes
20	25.00098	24.782	25.1558	Yes	24.983152	25.018805	25	Yes	25.00475	Yes
21	49.99913	49.8807	50.1417	Yes	49.978407	50.019854	50	Yes	50.0095	Yes
22	49.99909	49.8665	50.1423	Yes	49.977092	50.021086	50	Yes	50.009556	Yes
23	49.9988	49.8459	50.1605	Yes	49.975112	50.022486	50	Yes	50.014556	Yes
24	99.99618	99.8579	100.127	Yes	99.957791	100.03456	100	Yes	100.02989	Yes
25	99.99669	99.8601	100.132	Yes	99.955139	100.03823	100	Yes	100.02789	Yes
26	99.99635	99.8437	100.153	Yes	99.953223	100.03948	100	Yes	100.02589	Yes
19	24.9852	23.7926	26.2479	Yes	24.783955	25.186449	25	Yes	25.006472	Yes

Figure 6.34: Another Sample of Transmit Rate Validation Result of the 18-75-3 Simulation Scenario.

Message ID (base 10)	Model TX rate, arithmetic mean (ms)	Model TX rate, minimum (ms)	Model TX rate, maximum (ms)	Sufficient data points?	Model TX rate 95% percent confidence interval, lower bound (ms)	Model TX rate 95% percent confidence interval, upper bound (ms)	Nominal TX rate (ms)	Contained within confidence interval?	Test bench experiment TX rate (ms)	Contained within confidence interval?
5	6.250032	6.148218	6.342113	Yes	6.2482923	6.2517711	6.25	Yes	6.2510476	Yes
6	6.250118	6.150302	6.346377	Yes	6.2478051	6.2524313	6.25	Yes	6.2510272	Yes
7	6.250112	6.140071	6.36015	Yes	6.247217	6.2530068	6.25	Yes	6.2510272	Yes
8	6.250127	6.139823	6.359264	Yes	6.2466807	6.2535727	6.25	Yes	6.2510272	Yes
10	6.250145	6.14314	6.37086	Yes	6.2461044	6.2541848	6.25	Yes	6.2510272	Yes
1	6.250163	6.137505	6.38021	Yes	6.2458385	6.2544872	6.25	Yes	6.2510408	Yes
2	6.250203	6.045311	6.413537	Yes	6.2449825	6.2554227	6.25	Yes	6.2510408	Yes
3	6.250225	6.047976	6.404995	Yes	6.2448028	6.2556471	6.25	Yes	6.2510408	Yes
4	6.250255	6.043882	6.426364	Yes	6.2442433	6.2562666	6.25	Yes	6.2510408	Yes
9	6.250203	6.030877	6.433165	Yes	6.2436241	6.2567823	6.25	Yes	6.2512671	Yes
11	6.250246	6.039477	6.445039	Yes	6.2433884	6.2571027	6.25	Yes	6.251274	Yes
12	0	0	0	No	N/A	N/A	6.25	N/A	0	N/A
13	0	0	0	No	N/A	N/A	6.25	N/A	0	N/A
14	0	0	0	No	N/A	N/A	6.25	N/A	0	N/A
17	12.50009	12.3434	12.66879	Yes	12.4893	12.510876	12.5	Yes	12.502959	Yes
15	12.50017	12.29247	12.74351	Yes	12.488718	12.511617	12.5	Yes	12.502973	Yes
16	12.5002	12.28486	12.73826	Yes	12.488645	12.511753	12.5	Yes	12.502973	Yes
18	24.9995	24.77128	25.20613	Yes	24.981527	25.017465	25	Yes	25.004833	Yes
20	24.99958	24.86305	25.191	Yes	24.983274	25.015893	25	Yes	25.00475	Yes
21	50.00145	49.85774	50.20477	Yes	49.972864	50.030026	50	Yes	50.009556	Yes
22	50.00139	49.85516	50.21937	Yes	49.972569	50.030202	50	Yes	50.009556	Yes
23	50.00159	49.83846	50.21655	Yes	49.973121	50.030057	50	Yes	50.0145	Yes
24	100.0009	99.81346	100.1202	Yes	99.958679	100.04314	100	Yes	100.03067	Yes
25	100.0007	99.82152	100.146	Yes	99.958191	100.0432	100	Yes	100.02944	Yes
26	100.0005	99.832	100.1421	Yes	99.960455	100.04046	100	Yes	100.02767	Yes
19	24.98465	23.74996	26.29648	Yes	24.776543	25.192758	25	Yes	25.006917	Yes

Figure 6.35: Another Sample of Transmit Rate Validation Result of the 19-50-3 Simulation Scenario.

To explain how the transmit rate statistics from a single simulation run is used towards the overall results of the validation, the transmit rate statistics of the 18-75-3 simulation scenario shown in Figure 6.32 will be used as an example for discussion. The 18-75-3 simulation scenario transmits message Set 3, which has a total of 26 transmit messages. Since there is not enough data for 3 of the message IDs (Messages 12, 13 and 14), the total number of eligible message IDs for transmit rate

validation is 23. Out of the 23 eligible message IDs, 22 of them have a simulated transmit rate 95% confidence interval that contains their test bench transmit rates within the bounds (except for Message 24), and thus the number of containment for the test bench transmit rate is 21 in this particular simulation run. Since the nominal transmit rates of all eligible message IDs fall within their simulated transmit rate confidence interval, the number of containment for nominal transmit rate is 23. Similarly, the numbers of containment are counted for all 20 simulation runs of the 18-75-3 simulation scenario and they are listed in Figure 6.36, along with the number of eligible message IDs in each simulation run. Each row in the figure represent a simulation run, and it shows the numbers of containment for the nominal transmit rate in yellow background, the test bench transmit rates in blue background and the total number of eligible message IDs in red background. The totals of these numbers are summed and displayed in the row labeled "Total", and they will be later summed together with total numbers from other simulation scenarios towards the calculation of the overall results, the percentages of containment. Note that the percentages of containment of this particular simulation scenario are also shown in the chart but they are just for debug purposes (to check if the collected data are within expected range, *etc.*).

Number of simulation run	Number of containment for nominal TX rate	Number of containment for test bench TX rate	Number of eligible message IDs
1	23	23	23
2	23	23	23
3	23	23	23
4	23	23	23
5	23	23	23
6	23	23	23
7	23	23	23
8	23	23	23
9	23	23	23
10	23	23	23
11	23	22	23
12	23	23	23
13	23	21	23
14	23	22	23
15	23	23	23
16	23	23	23
17	23	23	23
18	23	23	23
19	23	23	23
20	23	23	23
Total:	460	456	460
Percentage of containment of current scenario:	100.000000%	99.130435%	

Figure 6.36: Intermediate Results: Numbers of Containment from All Simulation Runs of the 18-75-3 Simulation Scenario.

Once completed the 20 simulation runs for each simulation scenario, the results from all 14 simulation scenarios are collected into a chart shown in Figure 6.37. Each row in the middle that has a simulation scenario code in the first column shows the total numbers of containment and total number of eligible message IDs of that particular simulation scenario. To compute the overall results of the validation procedure, the totals of the two categories of number of containment from all 14 simulation scenarios are separately summed, and are shown in the row labeled "Overall number of

containment" near the bottom of the chart. The total of eligible message IDs from all 14 scenarios is also computed and is shown in the same row. Then, the overall percentage of containment for the nominal transmit rate is computed by dividing the overall number of containment by the total number of eligible messages IDs, and the same is performed for the test bench transmit rate. The overall percentages of containment are shown in the last row in Figure 6.37.

Simulation scenario	Total number of containment for nominal transmit rate	Total number of containment for test bench transmit rate	Total number of eligible message IDs
18-75-1	260	260	260
18-75-2	460	459	460
18-75-3	460	456	460
18-75-4	420	410	420
18-75-5	460	449	460
18-75-6	440	420	440
18-75-7	534	514	560
19-50-1	260	260	260
19-50-2	460	460	460
19-50-3	460	459	460
19-50-4	420	404	420
19-50-5	460	451	460
19-50-6	440	428	440
19-50-7	534	552	560
Overall number of containment (all scenarios combined)	6068	5982	6120
Overall percentage of containment (all scenarios combined)	99.150327%	97.745098%	

Figure 6.37: Transmit Message Transmit Rate Validation Overall Results: Percentage of Containment of the 95% Confidence Interval.

It appears that the overall percentage of containment of both nominal and test bench transmit rates are very close to 100%, and are at approximately 99.15% and 97.75%, respectively.

The overall percentage of containment for the test bench transmit rate is slightly lower than that for the nominal rate. There are 2 likely causes to this outcome:

- The message transmit rates profiled from the test bench traces (arithmetic means of the inter-arrival times) are generally slower (larger in value) than their nominal values, as can be seen in Figures 6.34 and 6.35, and this happens in all 14 test bench experiments. The test bench transmit rates vary randomly by message IDs, and vary for the same message ID in different experiment scenarios, which would introduce uncertainty into the comparison.

- As mentioned earlier, the timing effects from the host environment are modeled with pseudo-random number generation. Consequently, the model generated trace from each simulation run is unique, and thus will show a different set of message transmit rates that may have a higher or a lower number of containment.

The transmit rates of messages from simulations under the practical environment are very to their nominal and test bench experiment values. Thus it is concluded that the DFC simulation model is capable to transmit messages at the designated transmit rates in the practical (realistic) environment compared to the hardware test bench experiments.

6.2.5 Output Message Sequence on the CAN Bus

For validation of output message sequence on the bus compares the instances of the TX task, during which a message ID is transmitted. A message ID with "accurate output message sequence on the bus" during a simulation is transmitted in exactly the same instances of its TX task in the simulations compared to the corresponding test bench experiment that has identical DFC stack configuration. In order to make the comparison, each instance of the TX task is given an index (starting from 1,

from the start of the analysis) during a simulation, and the indices of the TX task instances corresponded to the transmissions of a message ID is collected into a list, for all messages IDs in the simulation. And the same process is performed for test bench experiment data as well. Figure 6.38 demonstrates how the "list of indices" of TX task instances is created for several message IDs in a hypothetical simulation scenario.

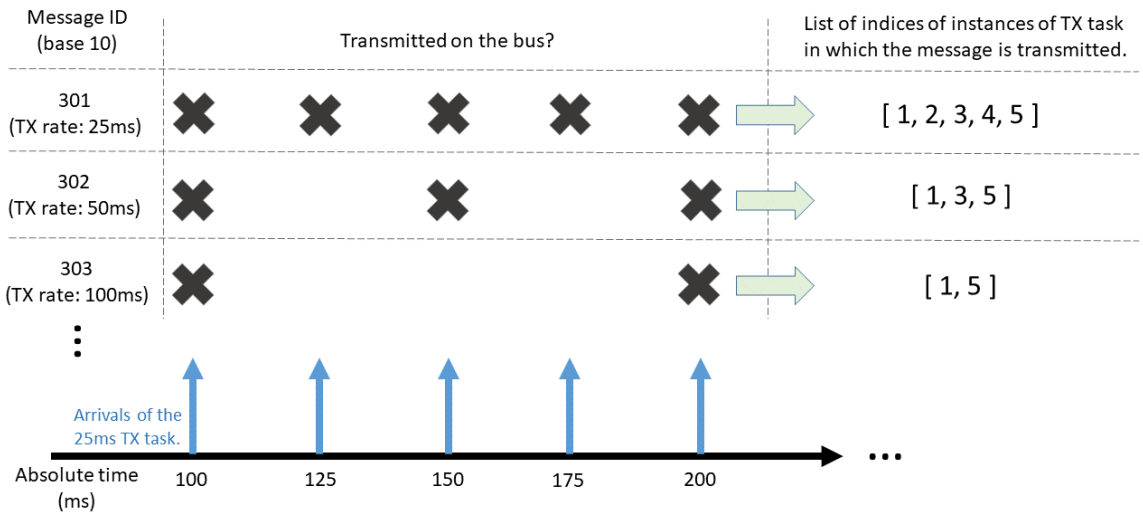

Figure 6.38: Creation of the List of Indices of TX Task Instances, in which the Message is Transmitted.

In this hypothetical simulation, the start of the first hyper-period is at simulation time 100ms. Therefore, for validation purposes, the first instance of interest of the 25ms TX task is the one that arrives at 100ms. The transmit messages shown in this example, Messages 301, 302 and 303, have transmit rates of 25ms, 50ms and 100ms, respectively, and they belong to the 25ms TX task. The 25ms TX task arrives at 100ms, 125ms, *etc.* Message 301 is transmitted in every single instance of the 25ms TX task from the start of the first hyper-period, and thus its list of indices is *[1, 2, 3, 4, 5]*. Message 302 is transmitted during every other 25ms TX task at a transmit

rate of 50ms, and thus its list of indices is *[1, 3, 5]*. Lastly, Message 303 has a rate of 100ms and therefore its list of indices is *[1, 5]*.

For validation of the output message sequence, for each simulation run, the entire model generated trace is profiled in order to identify all instances of the TX tasks, in which a message ID is transmitted, and the list of indices of those TX task instances are created for each message ID. The same profiling process is also performed for the test bench traces. Then, for each message ID, the list of indices from the particular simulation run is compared against the list of indices of the same message ID from the corresponding test bench experiment. If 2 lists of indices are identical for a message ID, then that message ID is counted as a message ID that has accurate output message sequence on the bus. One thing to note, since the test bench traces only contain 1 second of captured events while the simulations collect traces for 2 seconds, it is expected that the model generated traces capture more instances of TX tasks. Therefore, the comparison of the list of TX task instances is performed from the first to the last captured TX task in the test bench trace.

Similar to the validation of message transmit rates, the complete validation procedure for the output message sequences also consists of 20 simulation runs for each of the 14 simulation scenarios, and thus a total of 280 simulation runs are conducted. The validation procedure is demonstrated in Figure6.39.

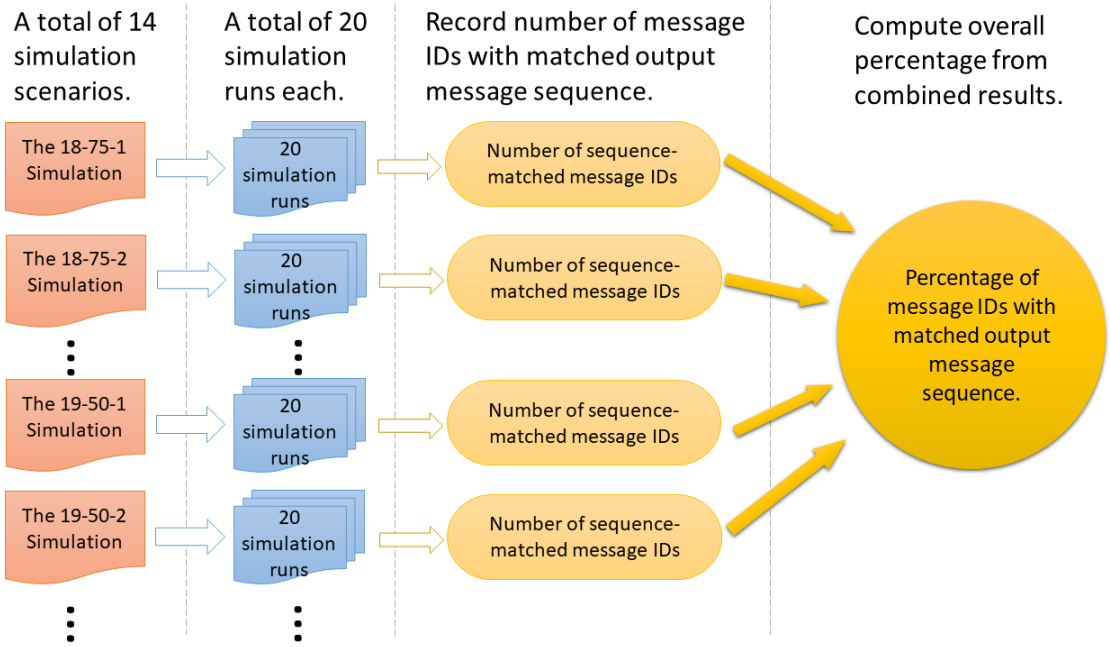

Figure 6.39: The Data Collection Process for Output Message Sequence Validation.

The total number of message IDs that have a matching output sequence ("sequence-matched message IDs") from each simulation run are summed together towards the overall number of sequence-matched message IDs. The overall result, the percentage of sequence-matched message IDs, is computed by dividing the overall number of sequence-matched message IDs by the total number of eligible message IDs throughout the validation procedure (message ID that appears on the bus at least once in a simulation run).

Message ID (base 10)	Individual output message sequence matched test bench trace?	Index of the first mismatched instance of TX task
5	Yes	N/A
6	Yes	N/A
7	Yes	N/A
8	Yes	N/A
10	Yes	N/A
1	Yes	N/A
2	Yes	N/A
3	Yes	N/A
4	Yes	N/A
9	Yes	N/A
11	Yes	N/A
12	N/A	N/A
13	N/A	N/A
14	N/A	N/A
17	Yes	N/A
15	Yes	N/A
16	Yes	N/A
18	Yes	N/A
20	Yes	N/A
21	Yes	N/A
22	Yes	N/A
23	Yes	N/A
24	Yes	N/A
25	Yes	N/A
26	Yes	N/A
19	Yes	N/A
Number of eligible message IDs	Number of message IDs with matched individual bus sequence	Percentage of sequence-matched message ID
23	23	100.000000%

Figure 6.40: Output Message Sequence Validation Statistics from a Single Simulation Run of the 19-50-3 Simulation Scenario.

Figure 6.40 above shows the statistics for output message sequence from a single simulation run of the 19-50-3 simulation scenario. The message IDs of the transmit message set is shown in the first column. The second column indicates whether the simulated output message sequence in this particular simulation run matches the output message sequence of the same message ID in the corresponding test bench

experiment. If a "N/A" is shown in this column, it means that this message ID has not shown up at all in the test bench, and the message ID in that row is then not eligible to be accounted the overall results. The third column shows the index of the first mismatch in the list of the TX task instances, if the output message sequences mismatch for message ID in that particular row. Otherwise a "N/A" is shown in the third column. In this particular simulation run, besides the 3 ineligible message IDs (Messages 12, 13 and 14, which never get transmitted due to limited maximum capacity of Queue 0), the output sequences in model generated traces of all messages are identical to their corresponding output sequences in the test bench trace, and consequently the entire third column shows "N/A". The 2 bottom rows show a summary of results for this simulation run, more specifically, the number of eligible message IDs, the number of sequence-matched message IDs and the percentage of sequence matched message IDs. In this particular simulation run, out of the 23 eligible message IDs, 23 message IDs have identical output message sequences on the bus compared to the corresponding test bench trace, which yields a 100% match.

As the durations of the timing effects from the host environment are modeled with pseudo-random number generation and their durations vary from one instance to another, it is possible to see mismatched output message sequences. Figure 6.41 below shows the simulation statistics from a single simulation run of the 18-75-7 simulation scenario, in which 2 message IDs have mismatched output message sequences compared to the test bench trace.

Message ID (base 10)	Individual output message sequence matched test bench trace?	Index of the first mismatched instance of TX task
5	Yes	N/A
6	Yes	N/A
1	Yes	N/A
2	Yes	N/A
3	Yes	N/A
4	Yes	N/A
7	Yes	N/A
8	Yes	N/A
10	Yes	N/A
17	Yes	N/A
18	Yes	N/A
20	Yes	N/A
21	Yes	N/A
22	Yes	N/A
23	Yes	N/A
24	Yes	N/A
25	Yes	N/A
26	Yes	N/A
28	Yes	N/A
9	Yes	N/A
11	Yes	N/A
12	Yes	N/A
13	Yes	N/A
14	Yes	N/A
15	Yes	N/A
16	Yes	N/A
19	No	1
27	No	11
Number of eligible message IDs	Number of message IDs with matched individual bus sequence	Percentage of sequence-matched message ID
28	26	92.86%

Figure 6.41: Output Message Sequence Validation Statistics from a Single Simulation Run of the 18-75-7 Simulation Scenario.

In this simulation run, Messages 19 and 27 (both belong to the 25ms TX task)

appear to have mismatched output sequences in this particular simulation run. The index of the first mismatched instance of the 25ms TX task is shown in the last column. The scheme for determination of the index of the first mismatch is shown in Figure 6.42, which demonstrates the cases of 2 hypothetical message IDs with mismatched output sequences on the bus.

Message ID (base 10)	Lists of TX task indices from both test bench and simulation trace		Comparison result
401	Test bench list of indices	[1, 3, 5, 7, 9, 11, 13, 15...]	Mismatch!
	Simulation list of indices	[1, 3, 5, 7, 9, 12, 14, 16...]	Index of first mismatched TX task: 11
402	Test bench list of indices	[1, 5, 9, 13, 17, 21, 26, 30, 34...]	Mismatch!
	Simulation list of indices	[1, 5, 9, 13, 17, 21, 25, 29, 33...]	Index of first mismatched TX task: 26

Figure 6.42: Determination of the Index of the First Mismatched Instance of the TX Task in Output Message Sequence Comparison.

When comparing between the 2 lists of indices, the test bench list is used as the reference. In the case of a mismatch, the index to be displayed in the simulation statistics chart is taken from the list of indices of the test bench trace. For Message 401, it is transmitted during the 12^{th} instance of its TX task in the simulation, while in the test bench experiment it is transmitted during the 11^{th} instance. Based on the pattern in the list of indices, Message 401 is mostly likely to become transmit-pending every 2 instances of the TX task (or in instance of the TX tasks with an odd-number index), and thus it should become transmit-pending during the 11^{th} instance. However, according to list of indices from the simulation, it is not transmitted until the 12^{th} instance. This could be caused by a full FIFO queue at the time of arrival of Message 401, and thus Message 401 is assessed again in a later instance of the TX

task. As a result, the comparison of the output sequences is a mismatch, and the index of the first mismatch shows "11". In the case of Message 402, it is transmitted during the 26^{th} instance of its TX task in the test bench experiment, but in the simulation it is transmitted during the 25^{th} instance. Different from Message 401, which is delayed in the simulation run, Message 402 seems to be delayed in the test bench experiment instead. However, since the test bench list is reference, "26" is shown in the simulation statistics as the index of the first mismatched instance of the TX task, instead of "25".

Once 20 simulation runs have been completed for a simulation scenario, the per-scenario results are gathered in a chart like the one shown in Figure 6.43, which shows the simulation statistics for all simulation runs of the 18-75-5 simulation scenario.

Number of simulation run	Number of message IDs with matched bus sequence	Number of eligible message IDs
1	22	23
2	22	23
3	22	23
4	22	23
5	22	23
6	22	23
7	22	23
8	22	23
9	22	23
10	22	23
11	22	23
12	22	23
13	22	23
14	22	23
15	22	23
16	22	23
17	22	23
18	22	23
19	22	23
20	22	23
Total:	440	460
Percentage of match of current scenario:	95.65%	

Figure 6.43: Output Message Sequence Validation Simulation Statistics of All Simulation Runs for the 18-75-5 Simulation Scenario.

The second column in Figure 6.43 shows the number of sequence-matched message IDs in each simulation run, and the third column shows the number of eligible message IDs. The summed totals are shown row labeled "Total" near the bottom of the chart.

Figure 6.44 shows the overall results for the output message sequence validation.

Simulation scenario	Total number of IDs with matched bus sequence	Total number of eligible message IDs
18-75-1	260	260
18-75-2	460	460
18-75-3	460	460
18-75-4	420	420
18-75-5	440	460
18-75-6	546	560
18-75-7	503	560
19-50-1	260	260
19-50-2	460	460
19-50-3	460	460
19-50-4	420	420
19-50-5	455	460
19-50-6	545	560
19-50-7	523	560
Overall number of sequence-matched IDs (all scenarios combined)	Overall number of eligible message IDs (all scenarios combined)	Overall percentage of message IDs with matched bus sequence
6212	6360	97.672956%

Figure 6.44: Overall Simulation Results for Output Message Sequence Validation.

The total number of sequence-matched message IDs from each simulation scenario is listed in the 2^{nd} (blue-background) column and the total number of eligible message IDs is listed in the 3^{rd} (red-background) column. The overall statistics are shown in the 2 bottom rows (green-background) of the chart, showing the overall numbers of both sequence-matched and eligible message IDs from all 14 simulation scenarios, and the overall percentage of the sequence-matched message IDs.

The simulation results show that a considerably high percentage, approximately 97.67%, of transmit message have identical output message sequences on the CAN network during the simulations compared to the corresponding test bench experiments. It can be concluded that, under the simulated practical environment, the DFC stack simulation model is able to transmit its messages in expected sequences according to the test bench experiment data.

6.3 Summary

In an ideal environment, where there is no randomness and no interference from the host environment, the DFC stack simulation model is able to transmit the periodic messages in expected sequences according to the specification documents. Under the practical environment, where the timing effects of the host environment are simulated through the use of statistical models based on the experiment data, the simulation model is able to transmit the messages at transmit rates that are acceptably close to their nominal transmit rates and profiled transmit rates observed from test bench experiments. Furthermore, the output message sequences of a considerably high percentage of messages are identical to the output message sequences observed in the test bench experiments. It can be concluded that the simulation model of the DFC stack is able to closely model the behaviors of the actual DFC stack running on the hardware test bench.

Although there are mismatches in the comparisons for the message transmit rates and output message sequences between the simulation and test bench results, the amount of mismatches is at an acceptable level and appears to be due to the existence of randomness in the simulated practical environment. The timing effects introduced by the host environment are modeled with pseudo-random number generation and thus their durations are expected to fluctuate. Furthermore, the flow of a message inside the DFC stack simulation model always involves not a single, but rather multiple modeled timing effect, which means that the fluctuation in the timely behaviors of each message transmission is affected by multiple pseudo-random number generations. As a result, even if there are mismatches in the comparisons of transmit rates and output message, the combination of the simulation model and the

simulated practical environment is capable of producing meaningful simulation data that close resembles the actual DFC stack running on the hardware test bench.

To look at the mismatches from a different perspective, since the statistical models of the timing effects are created based on the experiment results bench data, the simulations results obtained under the effects of the timing effects of the hardware test bench should be able to effectively predict practical but possibly unseen behaviors of the DFC stack running on the actual test bench. This can be further verified if more data is collected from the test bench, which, however will exceed the limitation of this thesis.

If comparing between the results from the simulations conducted under the ideal implementation and the simulated practical environment of the same DFC configuration, discrepancies can be observed in the timing behaviors such as message transmit rates. Figure 6.45 shows an example of discrepancies in the message transmit rates in a 19-50-5 simulation.

Message ID (base 10)	Practical TX rate 95% percent confidence interval, lower bound (ms)	Practical TX rate 95% percent confidence interval, upper bound (ms)	Ideal implementation TX rate, arithmetic mean (ms)	Contained within confidence interval?	Nominal TX rate (ms)	Contained within confidence interval?	Test bench experiment TX rate (ms)	Contained within confidence interval?
5	12.49804276	12.50185729	12.5	Yes	12.5	Yes	12.502641	Yes
6	12.49616192	12.5034509	12.5	Yes	12.5	Yes	12.502644	Yes
1	12.49577594	12.50386026	12.5	Yes	12.5	Yes	12.502699	Yes
2	12.49434799	12.50552912	12.5	Yes	12.5	Yes	12.502699	Yes
3	12.49359063	12.50624997	12.5	Yes	12.5	Yes	12.502699	Yes
4	12.49170702	12.50818457	12.5	Yes	12.5	Yes	12.502699	Yes
7	24.98978135	25.01026907	25	Yes	25	Yes	25.004222	Yes
8	24.9874344	25.01314772	25	Yes	25	Yes	25.005222	Yes
10	24.98662372	25.01377821	25	Yes	25	Yes	25.005917	Yes
17	74.97873549	75.02218478	75	Yes	75	Yes	75.010667	Yes
18	74.98120973	75.01989059	75	Yes	75	Yes	75.010667	Yes
20	74.98181195	75.01914478	75	Yes	75	Yes	75.010833	Yes
21	74.98299603	75.02484728	75	Yes	75	Yes	75.010917	Yes
22	74.98517035	75.02480922	75	Yes	75	Yes	75.010833	Yes
23	99.46354159	100.40896	99.930625	Yes	100	Yes	100.01444	Yes
24	99.46521932	100.4070048	99.930625	Yes	100	Yes	100.01544	Yes
25	99.46316414	100.4094013	99.930625	Yes	100	Yes	100.01544	Yes
26	99.46591668	100.4039547	99.930625	Yes	100	Yes	100.01533	Yes
28	99.45472596	100.402959	99.930625	Yes	100	Yes	100.01556	Yes
9	24.69388274	25.24212979	24.96686567	Yes	25	Yes	25.003917	Yes
11	24.69495386	25.24123639	24.96686567	Yes	25	Yes	25.003917	Yes
12	24.69598336	25.24071579	27.04918033	No	25	Yes	25.003889	Yes
13	24.69640829	25.24045219	27.04918033	No	25	Yes	25.003889	Yes
14	24.6966692	25.24034109	27.04918033	No	25	Yes	24.945829	Yes
15	24.78460446	27.49317747	27.04918033	Yes	25	Yes	26.559156	Yes
16	25.26707676	28.83220211	27.04918033	Yes	25	No	26.460818	Yes
19	74.65526171	75.34648391	100	No	75	Yes	74.990273	Yes
27	99.58316477	100.4000543	121.1709231	No	100	Yes	100.11138	Yes

Figure 6.45: Example of Discrepancies in Message Transmit Rates between Simulations Under The Ideal Implementation and The Simulated Host Environment.

The chart first shows the 95% confidence intervals of the simulated transmit rates under the practical environment, and they are compared to the simulated transmit rates in ideal implementation, nominal transmit rates and the test bench transmit rates. It appears that a relatively large number of ideal-implementation transmit rates are not contained within the confidence interval of the practical-environment transmit rates (compared to only 1 of the nominal transmit rate and 0 test bench rate

that are not contained). The difference between these ideal-implementation transmit rates and the nominal rates can be several orders of magnitude larger compared to that of the test bench rates: such as in the case of Message 19, the difference between the ideal-implementation and the nominal transmit rate is 25ms and there is only a difference of approximately 0,01ms for the test bench rates. Examples like this can be found in all simulation scenarios. Since the simulation model with the simulated host environment has been validated in terms of closely resembling the behaviors of the actual system (i.e. practical-environment transmit rates resembles test bench transmit rates), these discrepancies effectively demonstrate the improvement in the fidelity of the simulation results when having the details of the actual system incorporated into the implementation of the simulation model.

Chapter 7

Conclusions

In the ideal implementation where there is no randomness and interference from the host environment, the simulation model of the DFC stack is capable to enqueue and load messages into TX buffers in expected orders according to the specification documents. With the help of the method proposed in this thesis, which aims to identify and incorporates the system details into the implementation of the simulation model, it is possible to run simulations under a simulated practical environment, where the timing effects introduced by the host environment (the combination of the RTOS and the dual-core ECU test bench) are simulated through the use of statistical models of their duration data. And the simulations under the simulated practical environment have proved that the simulation model is capable of transmitting the periodic messages at rates that are close to the nominal and test-bench-experiment transmit rates, and that the output message sequences on the CAN bus during the simulations closely resembles the output sequences of the actual test bench experiments. It can be concluded that the combination of the simulation model and the method used in this research is capable of generating realistic simulation results that closely reflect the

behaviors of the actual system. This should provide a high level of confidence to the user of this simulation model that the model is a possible substitute of the hardware test bench setup that also offers additional possibilities, such as vast diversity of test scenarios or different purposes at various stages of development cycle.

7.1 Future Research

Even though very good results have been obtained during the validation of the simulation model, mismatches have occurred in the comparisons of message transmit rates and output message sequences during simulation. We believe these mismatches are due to the use of random number generation (statistical models) during the simulations. It is possible that these mismatches are signs of behaviors that only occur infrequently but did not appear in the limited amount experiment data available for this research. This can be validated if more experiment results are available, more specifically, more data from each test bench experiment scenario, or data from experiments with a larger variety of combination of DFC stack calibration values. If successfully validated, the simulation model can be used to conveniently predict corner cases that occur at very low possibility and require a large amount of experimentation on a test bench.

Future research could also focus on further validation of the model in additional application scenarios. The DFC stack model could also be integrated with the Vector CAN stack model of Whinton (2016) to allow simulation of networks with ECUs utilizing the different CAN stacks as often occurs in production vehicles. Further investigation of how the combined simulation models could be used in the network design workflow of engineers working on new vehicle designs or trying to validate fixes

for issues that have been identified in existing networks.

Bibliography

Cervin, A., Henriksson, D., and Ohlin, M. (2010). *TrueTime 2.0 beta - Reference Manual*. Lund University.

Di Natale, M. (2009). Controller area network. URL: `http://retis.sssup.it/~marco/files/lesson9-controller_area_network.pdf`.

Di Natale, M., Zeng, H., Giusto, P., and Ghosal, A. (2012). *Understanding and Using the Controller Area Network Communication Protocol: Theory and Practice*. Springer Science & Business Media.

Hao, J., Wu, J., and Guo, C. (2011). Modeling and Simulation of CAN Network Based on OPNET. In *IEEE 3rd International Conference on Communication Software and Networks (ICCSN), 2011*, pages 577–581. IEEE.

Herpel, T., Hielscher, K., Klehmet, U., and German, R. (2009). Stochastic and Deterministic Performance Evaluation of Automotive CAN Communication. *Computer Networks*, 53:1171–1185.

Hofstee, J. and Goense, D. (1999). Simulation of a Controller Area Network-based Tractor - Implement Data Bus according to ISO 11783. *Journal of Agricultural Engineering Research*, 73:383–394.

Jiang, S. and Zhang, Y. (2016). Method and apparatus for fault detection in a controller area network. URL: `https://www.google.com/patents/US9524222`.

Kaufer, M. L., Risse, P. L., and Crites, T. A. (2015). Message transmission control systems and methods. URL: `https://patents.google.com/patent/US9002533B2`.

MathWorks (2014). *Documentation/SimEvents/SimEvents Examples*.

Matsumura, J., Matsubara, Y., Takada, H., Oi, M., Toyoshima, M., and Iwai, A. (2013). A Simulation Environment Based on OMNeT++ for Automotive CAN-Ethernet Networks. In *4th International Workshop on Analysis Tools and Methodologies for Embedded and Real-time Systems (WATERS2013)*, pages 1–6.

Prodanov, W., Valle, M., and Buzas, R. (2009). A Controller Area Network Bus Transceiver Behavioral Model for Network Design and Simulation. *IEEE Transactions on Industrial Electronics*, 56:3762–3771.

STMicroelectronics (2017). *STM32F4 Reference Manual*. STMicroelectornics.

Vector Informatik GmbH (2014). *Vector: CANoe & Modules*.

Whinton, G. (2016). Higher-fidelity Modelling and Simulation of the CAN Protocol Stack. Master's thesis, McMaster University.