High Availability MilCAN

by Panagiotis Ioannis Oikonomidis

Submitted in the fulfilment of the
requirements for the degree of
Doctor of Philosophy

School of Science and Technology
Department of Engineering and Design
University of Sussex, Brighton UK
September 2010

To my sweet mother Elisabeth

Declaration

I hereby certify that the attached thesis is my own work, except where indicated. I have identified my resources and in particular I have put in quotation marks any passages stated word for word and identified their origins. I also declare that this thesis has not been submitted, either in the same or different form, to this or any other University for a degree.

The VSI GUI mentioned in chapter 4 and 6 was developed by G. Valsamakis. The MilCAN Reconfiguration protocol used for the communication of the bootloader with the VSI GUI was a combined work between the author of this thesis and G. Valsamakis. The VSI Bridge used in the VSI testbed mentioned in chapter 6, was developed by P. Charchalakis.

Panagiotis Ioannis Oikonomidis - 2010

Abstract

Modern vehicles incorporate an increasing number of electronic systems. These systems consist of multiple embedded devices in order to replace already existing mechanical or hydraulic solutions or introduce new functionalities. The demand of using electronic systems for the military land vehicles is continuously increasing and much higher than in the commercial sector. For these electronic devices to intercommunicate and provide the desired service, embedded networks have to be used.

These networks need to be compliant to the minimum requirements set by their intended use. Some of these requirements are determinism, high bandwidth, flexibility, durability and cost. There are networks that provide some of the above mentioned characteristics, but rarely all of them. One solution for this; is to improve an already existing embedded network to provide the extra desired functionality and also keep the system complexity minimal.

This study represents an investigation to expand the functionalities to an existing military protocol by adding reconfiguration for Through Life Capability Management (TLCM) and fault-tolerant capabilities for high operational system availability. The network that is used for this research is MilCAN which is software based protocol based on CAN. MilCAN stands for Military CAN and is a deterministic data transfer protocol used in control systems. For this research MilCAN A is used which uses 29bit Identifier and allows both periodic and event driven data to be transmitted. These extra functionalities need to be software based to provide flexibility and upgradability without hardware restrictions. The devices with the added capabilities need to be compatible with older devices, to provide the flexibility to the system designer to choose when he will use them. To verify the operation and performance of the added functionalities two testbeds have been developed, the first testbed is used for development and operation verification where the second testbed is used for operation verification and performance measurements while emulating the operation of a vehicle.

The output of this research is accepted by the MilCAN Working Group (MWG) as an addition to the MilCAN protocol specifications. The MWG was formed in 1999 when a need was recognised to standardise the implementation of CANbus within the military vehicles community. The additional functionalities are included as optional enhancements in the MilCAN protocol specifications which gives the system designer the flexibility to decide depending his system requirements.

Acknowledgements

I would like to thank my supervisor Dr Elias Stipidis. I am grateful for his directness, his ideas and encouragement during this project. Thank you for showing patience all these years and providing me with both professional and personal guidance.

Personally, I would also like to thank my second supervisor Dr Falah Ali and my research colleagues and friends at the Vetronics Research Centre; Dr P. Charchalakis, Dr G. Valsamakis, J. Melentis, Dr O. Obi, Dr I. Colwill, Dr M. Fowler, D. Summers, T. Webber, S. Gabrovsek, D. Abdulmasih, A. Despande and G. Thomeczek.

Additionally I would like to thank my friends that provided me with moral support during the difficult times A. Kimoundris and G. Mamalis.

I owe my deepest gratitude to my family, especially my mother Elisabeth Oikonomidou and my late fathers Konstantinos Oikonomidis and Dr Alexantros Tsokos for guiding me and made this possible. This thesis would not have been possible without you. Also I would like to thank my sweet fiancée Konstantina Stamogianni for her faithful support and patience during this work.

Finally I would like to acknowledge the UK MoD (Ministry of Defence) and MilCAN Working Group for their financial support and feedback in completing this work.

Contents

List of Figures

List of Tables

List of Acronyms & Abbreviations

µC	Microcontroller
AIVF	Application Interrupt Vector Forwarders
API	Application Programming Interface
BI	Babbling Idiot
BIT	Build In Test
BIVF	Bootloader Interrupt Vector Forwarders
BST	Bosch Siemens Temic
CAN	Controller Area Network
CNI	Communication Network Interface
D2B	Domestic Digital Data Bus
DSI	Distributed Systems Interface
EML	Error Management Logic
EMU	Engine Management Units
EPA	Enhanced Performance Architecture
EWRN	Error Warning Status
FIP	Factory Instrumentation Protocol
FMEA	Failure Modes and Effects Analysis
FT	Fault Tolerance
GUI	Graphical User Interface
HRT	Hard Real Time
HUMS	Health and Usage Monitoring Systems
IE	Inter Equipment
IEC	International Electrotechnical Commission
IEEE	Electronic and Electrical Engineering
IHSDB-UG	International High Speed Data Bus Users Group
ISA	Instrumentation, Systems, and Automation society
LIN	Local Interconnect Network

LLC	Logical Link Control
MAC	Medium Access Control
MEDL	Message Descriptor List
MI	Motorola Interconnect
MML	Mobile Multimedia Link
MOST	Media Oriented Systems Transport
MWG	MilCAN Working Group
NMS	Network Management System
NRT	Non Real Time
OBD2	On-Board Diagnostics II
OSI	Open Systems Interconnection
PLC	Programmable Logic Controllers
PTU	Primary Time Unit
PWM	Pulse Width Modulated
REC	Receive Error Counter
SRT	Soft Real Time
STP	Shielded Twisted Pair
TDMA	Time Division Multiple Access
TEC	Transmit Error Counter
TLCM	Trough Life Capability Management
TTP	Time Triggered Protocol
UTP	Unshielded Twisted Pair
VPW	Variable Pulse Width
VRC	Vetronics Research Centre
VSI	Vetronics System Integration
WBS	Weighted Bus Selection

Chapter 1 Introduction

1.1 Prologue

Modern vehicles incorporate an increasing number of electronic systems. The demand of using electronic systems for the military vehicles is increasing over the years. The vehicles are fitted with various systems to provide additional functionalities to the driver and advantages in electronic warfare. In passenger vehicles, the electronic systems to assist the driver are becoming more popular over the years.

These electronic systems require inter-communication between their components. Depending the use of the system there are different requirements such as determinism, high bandwidth, flexibility, durability and cost. Because different networks fulfil different requirements, in a system, there may be more than one type of network available. This affects the cost of the system. For example weapon control systems require safety critical systems and sensors require deterministic systems.

The objective of this research is to enhance the network availability and the Through Life Capability Management (TLCM), of an already existing low cost protocol targeted for military vehicles. Availability refers to the property of the system or a system resource being accessible and usable upon demand by an authorized system entity, according to performance specifications for the system [Stallings'06]. Reliability is defined as the ability of a system or component to perform its required functions under stated conditions for a specified period of time [Iee'90]. The protocol that is used is a real time CAN based; called MilCAN. MilCAN stands for Military CAN and is a software based non-fault tolerant deterministic protocol. There are two variations of MilCAN, MilCAN A and MilCAN B. For this research MilCAN A is used, which uses a 29 bit Identifier compared to MilCAN B which uses an 11 bit identifier. MilCAN on its own is not suitable for applications that have safety integrity requirements. MilCAN is best suited for control applications that require determinism and not safety critical.

The first functionality that is added is MilCAN Reconfiguration, which allows the system maintainer to have more control and variety of actions. The second functionality is the MilCAN Fault Tolerance, which allows the MilCAN network to continue normal operation for longer when errors and faults are introduced into the system as a result increasing the availability of the system. Fault tolerance is the attribute that enables a system to achieve fault-tolerant operation and a fault-tolerant system is one that can continue to correctly perform its specified tasks in the presence of hardware failures and software errors [Johnson'88]. The development of the MilCAN additions was a two way process. Firstly, they were developed on a one segment desktop testbed; using of the self-test equipment. Secondly, on a three segment testbed designed to emulate the operation of a vehicle.

The objectives of the research presented in this thesis are to add the two extra functionalities to the already existing MilCAN protocol trying to address its shortfalls. The requirements for the projects are:

- Through Life Capability Management
- Increase the network availability
- Easy configuration procedure
- Node compatibility between Fault Tolerant and non-Fault Tolerant Devices
- Hardware independent operation
- Flexible MilCAN bus configuration
- Fault recovery

The output of this research is both theoretical and practical development. The MilCAN protocol specifications are vital for this research, thus studied in detail. Key missing characteristics are identified and contributed to provide the targeted additional functionalities. Additionally, the output of this research is provided to the MilCAN working group, as an addition to the MilCAN specification.

The MilCAN reconfiguration is developed to be able to remotely reconfigure MilCAN devices. This is achieved by using a custom bootloader which is hardware platform dependant and was developed to provide all the desired reconfiguration capabilities. To control the bootloader the VSI Graphical User Interface (GUI) software is used, which provides to the user the means of controlling the bootloader’s operations. The communication of the device’s internal bootloader with the VSI GUI is achieved through a newly developed protocol; that ensures uninterrupted execution of the reconfiguration operations.

The MilCAN Fault Tolerance Layer is developed to be able to withstand a big variety of faults and errors introduced into the system. The developed algorithm is coded between the MilCAN layer and the Application layer to provide seamlessly multi bus support. Additionally, it provides advanced error detection and bus recovery techniques, which are not currently present in the MilCAN protocol. The communication between the Fault Tolerant Layer of different devices is established again with a newly developed protocol; that ensures the synchronised operation of all Fault Tolerant devices. Additionally, to closely monitor the operation of the FT Layer, an internal monitoring system is developed, which works in collaboration with a monitoring computer.

Evaluation and verification of the High Availability MilCAN is achieved through two testbeds. The first testbed includes one MilCAN segment connected to a desktop computer, which provides a controlled environment appropriate to be used during the development and later to verify the operation of the system. The testbed is constructed by using Commercial Off-the-Shelf Equipment (COTS). The second testbed includes three MilCAN segments and a pair of MilCAN and Ethernet backbones. This testbed emulates a real-life military vehicle and provides the means to conduct certain operation and latency tests. The second testbed is also constructed with COTS.

1.2 Previous work at the Vetronics Research Centre

The Vetronics Research Centre (VRC) is part of the Communications department of University of Sussex. Previous work developed at the VRC used in this research is the Vetronics System Integration (VSI) Bridge and the VSI Graphical User Interface (GUI).

The VSI Bridge is a highly configurable bridge/router, which interconnects different MilCAN segments and evaluates their operation. It is designed to interconnect the MilCAN segments through a pair of MilCAN and Ethernet backbones. The functionalities that it provides are configurable routing operations, remote monitoring and management of the internal system and the attached MilCAN segments. The VSI Bridge is described in detail in Dr Charchalakis thesis [Charchalakis'05].

The VSI GUI provides the configuration capabilities to the VSI Bridge. It connects to the VSI Bridge through the Ethernet backbones, providing an easy way to configure and maintain the VSI Bridge. The VSI GUI had to be modified in order to provide the extra functionalities required by the High Availability MilCAN. More details can be found in Dr Valsamakis thesis [Valsamakis'06].

The MilCAN segments that are interconnected by the VSI Bridges were developed and constructed during the author's master's project. The microcontrollers that were initially used were C167CR offering a single CAN bus. During the author's PhD the microcontrollers were upgraded to the C167CS that offer dual CAN bus, which are suitable for this research.

1.3 Thesis layout

Chapter 2 provides an overview on fieldbuses used in the industry for control and automation and focus in their use in vehicle electronics. A brief history of the fieldbuses is presented followed by the existing networks available and focusing more on the vehicle oriented ones. Furthermore the architecture of a couple of fault tolerant enabled networks is briefly analysed.

Chapter 3 describes the specifications of the two protocols CAN and MilCAN. The research work presented in this thesis is based on the MilCAN protocol. It is very important for the reader to understand the protocol in order to recognise the limitations that this thesis is trying to address. Additionally the output of this research is provided to the MilCAN working group, as proposed additions for the MilCAN specification.

Chapter 4 is the first contribution chapter. It describes in detail the MilCAN Reconfiguration and its components. The operation procedure of the MilCAN Reconfiguration is also described.

Chapter 5 is the second contribution chapter. It presents the MilCAN Fault Tolerant Layer which is part of the High Availability MilCAN. Again in this chapter the components and operation of the layer are presented.

Chapter 6 is the third contribution chapter presenting the performance and operation of the High Availability MilCAN components. The evaluation of the components is established with the use of the appropriate vetronics testbeds. The design and layout of the testbeds are discussed, along with the test scenarios followed.

Chapter 7 concludes the thesis with the discussion of the research work contained in the previous chapters, the lessons learned during the development and potential future work.

Chapter 2 Vetronic networks

2.1 Introduction

Fieldbuses are widely used in embedded systems where their main applications are industrial and automation control. Main advantage of the fieldbuses is their capability of multiple devices to be integrated on a single network which reduces the topology complexity and cost of design, build and maintenance of the network. Fieldbuses simplify the task of modifying the network which makes future expansions less troublesome. Vehicles have an increasing number of electronic devices that need to be integrated together, and the fieldbuses are the main candidates for this use.

This chapter introduces the fieldbus technology and its use in the automotive industry as a real-time communication network. Initially the history of fieldbuses is presented and their real world application uses, followed by the actual fieldbuses that are used in different applications. A detailed view on commonly used fault tolerant architectures is provided.

2.2 Fieldbuses history

The first fieldbus created was the Factory Instrumentation Protocol (FIP) fieldbus in mid 1980s by the French Italian FIP club. The second were the Germans with the Profibus which is based on the Programmable Logic Controllers (PLC). Third fieldbus was the Foundation Fieldbus developed by the United States. Since then there have been more and more fieldbuses been developed in order to allow different devices to communicate with each other.

Because there was an increasing number of fieldbuses there was a need for standardisation of the specifications and conformance test-suites to verify the device compliance to the specifications. Standardisation organisations such as the Institute of Electronic and Electrical Engineering (IEEE) and Instrumentation, Systems, and Automation society (ISA) in the United States, Cenelec in Europe, and International Electrotechnical Commission (IEC) worldwide, assisted to the standardisation [Pinceti'04].

2.3 Fieldbuses networks

There is a large number of fieldbuses being developed by various companies since there is a need in industrial automation and control, vehicle electronics networks to connect field devices such as drive controllers, sensors, regulators and more [Pierre Thomesse'99]. With the use of fieldbuses the use of wires to connect the devices is reduced and there are individual applications that communicate with a common protocol [Glanzer'96].

Distributed systems require real-time communication capabilities, which are offered by a range of fieldbuses. Since there is a wide range of fieldbuses the selection of the right protocol depends on the intended operation of the developed system. The following criteria distinguish the various options available [Patzke'98]:

- Physical data transmission quality.
- Data coding and transmission type.
- Physical communication channel sharing.

The typical characteristics of a fieldbus protocol are [Schumny'98a]:

- Serial bus topology with multi-point interfaces.
- Software controlled interface management and data transfers.
- Data transfers could be:
  - Asynchronous and code-dependent.
  - Synchronous and code-dependent or independent.
- Network layer and presentation layer functions are merged with data-link and application layer functions respectively.
- Channel access and transmission control functions are based on the application requirements, such as real-time operation, and in case of cyclic sampling of digitised data they must fulfil the Shannon Theorem.

2.4 Fieldbuses structure

According to the Open Systems Interconnection (OSI) model the fieldbuses have a layered structure as can be seen in table 2.1 [Zimmermann'80]. Data from each layer are passed to layer below that encapsulates them within their own data starting from the top layer. The layered approach provides flexibility into the ways a packet could be processed but the data packet sizes and the processing required from the devices affects the real-time operation of the network when using embedded devices with limited resources. Because of these overheads the fieldbuses moved from the OSI model to the Enhanced Performance Architecture (EPA) model which has only three layers as can be seen in table 2.2 [Schumny'98a]. Although the layers have been reduced from seven (OSI) to three (EPA), functionalities have not been discarded but absorbed by the Application and Data Link layers. This way overall system latencies are reduced since the protocol organisation is more efficient with less processing overheads imposed to the application.

Table 2.1: Open Systems Interconnection (OSI) model

Layer #	Description
7	Application layer
6	Presentation layer
5	Session layer
4	Transport layer
3	Network layer
2	Data Link layer
1	Physical layer

Table 2.2: Enhanced Performance Architecture (EPA) model

Layer #	Description	Functionality
3	Application layer	Complete definition of functions to guarantee communication between partners. Example: Real-time or random access; cyclic or interrupt driven transport; time-variant or invariant field signals; samples or message oriented signals.
2	Data Link layer	Channel Access; Medium Access Control (MAC); synchronisation and coding; Logical Link Control (LLC); error control. Example: Deterministic (e.g. polling, token-passing) or probabilistic access (e.g. CSMA/CD); synchronous or asynchronous character transfer; code-dependent or independent(transparent) approach; hardware or software control.
1	Physical layer	Organising of bit-stream transfer by definition of electrical characteristics, connectors, and topology of the system. Example: Balanced or unbalanced electrics with or without galvanic, shielded or unshielded; point-to-point or multipoint topology with parallel or serial character transfer, interface close to processor or to peripherals.

2.5 Automotive fieldbus networks

Modern vehicles incorporate an increasing number of electronic systems. The typical mechanical and hydraulic solutions for braking and steering for example, are now replaced by electronic systems controlled by software. There are various systems to assist the vehicle operation such as ABS for intelligent breaking, Engine Management Units (EMU) for the engine, air-bags for passenger safety, GPS for driver assist and sound and video player for entertainment [Powers'00].

Since the electronic devices used in a vehicle have increased over the years, the cabling also increased from 45 meters used in 1955 to 4km at present time. When using networks to interconnect the devices less wires are required, this simplifies the layout of the system and reduces the weight of the car. In figure 2.1 a modern vehicle that uses multiple networks for its sub-systems. There are many different vehicle buses available but currently the most popular one is CAN.

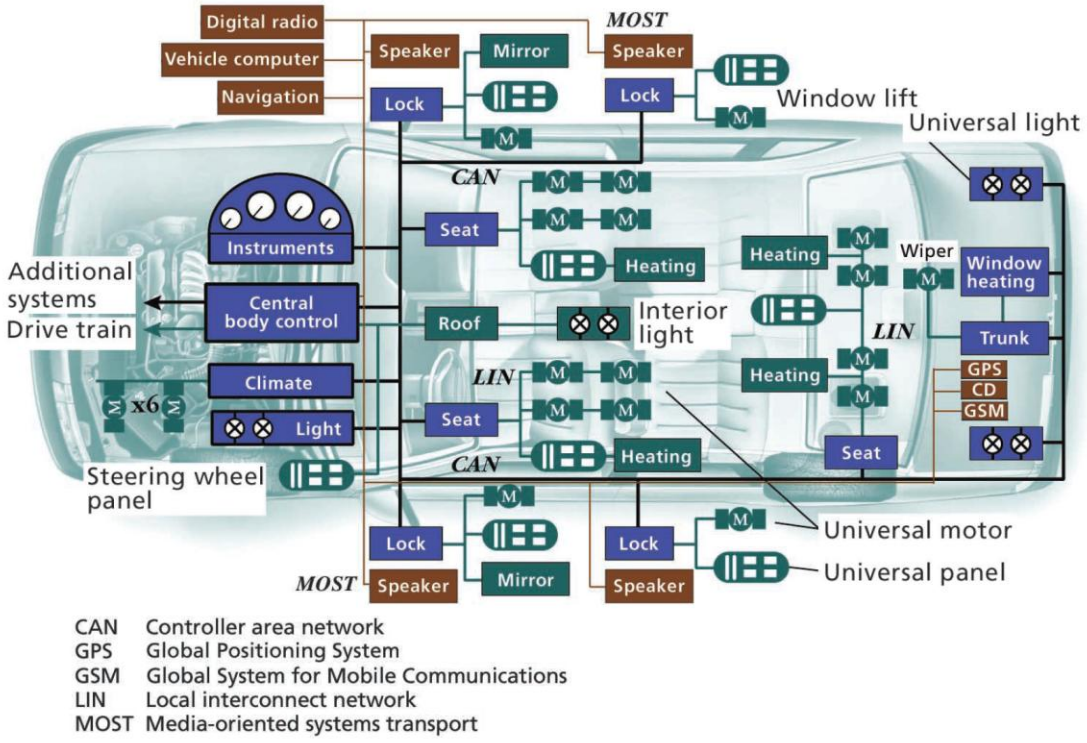


Figure 2.1: Modern vehicle's network architecture [Leen'02]

2.6 Vehicle bus technologies

Further classification to vehicle buses can be done based on the targeted application areas of their technologies. These groups are [Charchalakis'05]:

- **Multimedia buses** High speed networks for devices serving multi-media content.
- **Diagnostics buses** Standardised protocols to provide diagnostics to the internal devices of the vehicle.
- **Safety-critical buses** Highly deterministic, real-time, and fault-tolerant networks used for interconnection of devices associated in systems that control the vehicle and can thus affect the safety of the passengers.
- **Sub-system buses** Interconnecting the main systems of the vehicle.
- **Consumer buses** Proprietary networks that have been developed for specific applications.

2.6.1 Multimedia buses

MOST The Media Oriented Systems Transport (MOST) bus is designed for multimedia applications within the vehicle that require high speed data transfers. It is based on point-to-point connections in ring, star, or daisy-chain topology using fiber-optic links. Within the MOST specifications all the associated layers are specified, such as the Physical, Medium Access Control, Network, and Application layers. Transmission and reception of data is done by converting TTL signals to and from option, using appropriate converters.

MML Bus The Mobile Multimedia Link (MML) Bus is a high-speed multimedia bus structured as a star topology through optical fiber. It's a fault tolerant Master and Slave bus with 100Mbps bandwidth and plug-and-plat capabilities.

D2B bus The Domestic Digital Data Bus (D2B) is a high-speed optical multimedia bus for audio, video, and data applications. Structured as a ring or star topology it offers speeds up to 20Mbps with a maximum bus length of 10 meters.

SMARTwireX The SMARTwireX protocol is an addition physical layer for D2B buses to provide up to 25Mbps bandwidth over standard UTP cables within the EMC limits for automotive. It also extends the maximum bus length up to 150 meters.

2.6.2 Diagnostics buses

SAE J1850 The SAE J1850 protocol is used as a diagnostics and application data bus. It supports two physical layers, a Pulse Width Modulated (PWM) differential connection with 41.5Kbps bandwidth, and a Variable Pulse Width (VPW) connection with 10.4Kbps bandwidth.

OBD2 The On-Board Diagnostics II (OBD2) bus is a diagnostics protocol for vehicles with a predefined standard connection. It is used as a standardised protocol to monitor vehicle emissions of passenger cars.

SAE J1939 The SAE J1939 is a protocol used in communication and diagnostics within heavy duty trucks. It is based on CAN2.0b and operates with a customised arbitration field. The specification contains standardised message sets with identifiers assigned to specific functions that should be supported by compliant devices. [Fredriksson'02]

2.6.3 Safety-critical buses

TTP The TTP bus is a time-triggered protocol developed for highly deterministic real-time applications with error detection and fault tolerance requirements.

ByteFlight The ByteFlight bus is a high-speed bus for safety-critical applications. It uses Time Division Multiple Access (TDMA) over 2/3 wire optical-fibre lines to provide 10Mbps bandwidth. Available topologies are Start and Cluster with an information update rate of 250uS. The Master and Slave configuration using message-oriented transmission allows broadcasting of data to multiple devices simultaneously. [Hammett'03]

FlexRay The FlexRay bus, an extension of ByteFlight, is a high-speed serial communication bus over Unshielded or Shielded Twisted Pair (UTP/STP). A point-to-point or star topology offers 10Mbps bandwidth with time-triggered operation making it fault-tolerant and deterministic. FlexRay was designed for safety-critical applications such as steer-by-wire and brake-by-wire.

DSI Bus The Distributed Systems Interface (DSI) bus is another protocol developed by Motorola for safety applications. It provides a two wire serial communication between sensors and safety actuators such as air-bags. The connection is between Master and Slave devices at 150Kbits with a 4bit CRC.

BST Bus The Bosch-Siemens-Temic (BST) bus is a safety bus with a two-wire connection at 250Kbps using Manchester encoding with either Parity or CRC error correction and detection.

Intellibus The Intellibus bus is similar to CAN bus, originally developed for military avionics. It supports higher speeds up to 12.5Mbps and is used in the automotive area for drive-by-wire applications. With a Master and Slave multi-drop topology it supports up to 30 meter UTP/STP cables with 64 nodes at 12.5Mbps. As a safety-critical bus it provides parity and CRC error detection.

2.6.4 Sub-system buses

MI Bus The Motorola Interconnect (MI) Bus is a single wire serial communication protocol supporting a single master and multiple slave devices. It’s a simple protocol for basic low-speed networked applications such mirror control, windows control, etc.

IE Bus The Inter Equipment (IE) bus is a low-speed half duplex asynchronous communication protocol. With two operational modes, one at 3.9kbps and one at 17kbps or 18kbps, depending on the oscillator frequency, up to 50 nodes can be connected on a bus with a maximum length of 50 meters.

LIN Bus The Local Interconnect Network (LIN) is a simple serial bus for interconnection between sensors and actuators. It supports speeds up to 19.2kbps over a maximum bus length of 40 meters.

CAN bus The Controller Area Network (CAN) bus is high-speed bus used in a wide range sub-system applications. It’s an asynchronous broadcast communication bus supporting up to 1Mbps bandwidth.

SAE J1708 The SAE J1708 is a serial bus for communication between microcomputer systems within a heavy-duty vehicle. Based on the RS-485 electrical layer, it provides 9.6Kbps bandwidth with a cable up to 40 meters.

2.6.5 Consumer buses

IDB-1394 The IDB-1394 is an implementation of the Firewire (IEEE 1394) for the automotive. It provides high-speed serial communication over UTP multidrop cables, based on the CAN 2.0B physical layer, for consumer applications.

2.7 Fault tolerant architectures

The reason TTP and FlexRay are used for safety critical applications, is because both are fault tolerant, deterministic and high bandwidth.

2.7.1 TTP/C

TTP/C is a time-triggered communication protocol for safety critical distributed real-time control systems. As a time-triggered protocol the messages are transmitted periodically.

2.7.1.1 Network Structure

The TTP/C network consists of electronic modules (nodes) interconnected by two separate broadcast busses which can have speeds up to 10Mb/s. These two buses are called channel 0 or channel A and channel 1 or channel B.

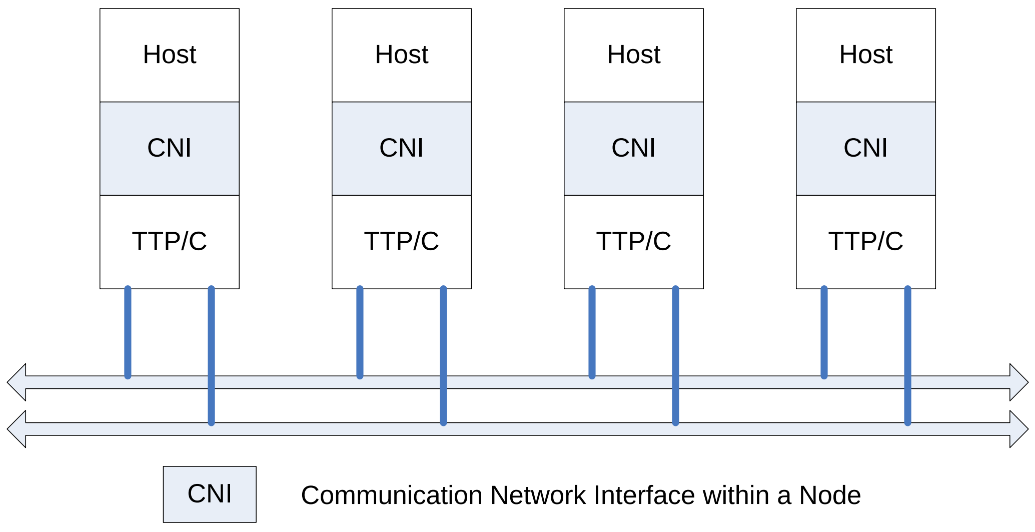


Figure 2.2: Example of TTP/C network

The Communication Network Interface (CNI) consists of a memory area that allows simultaneous random access for the host CPU and the TTP/C controller figure 2.2. The TTP/C can operate in different network topologies. The most common are the bus topology as seen in figure 2.3, star topology figure 2.4 and multi-star topology figure 2.5. The star topology uses central star-coupler devices which operate as a central bus guardian and provide a better fault tolerant network.

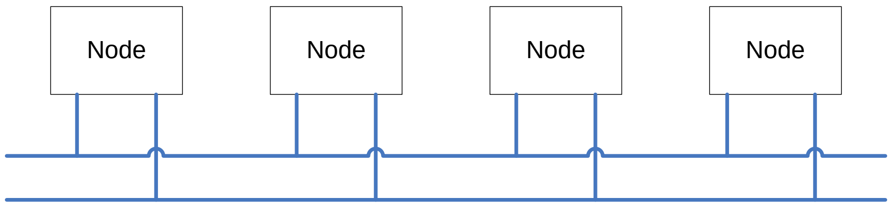


Figure 2.3: TTP/C bus topology

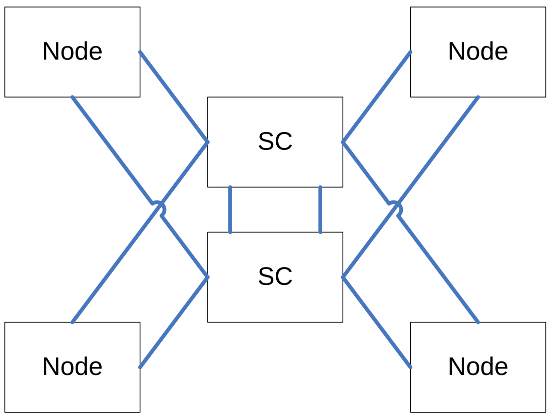


Figure 2.4: TTP/C star topology

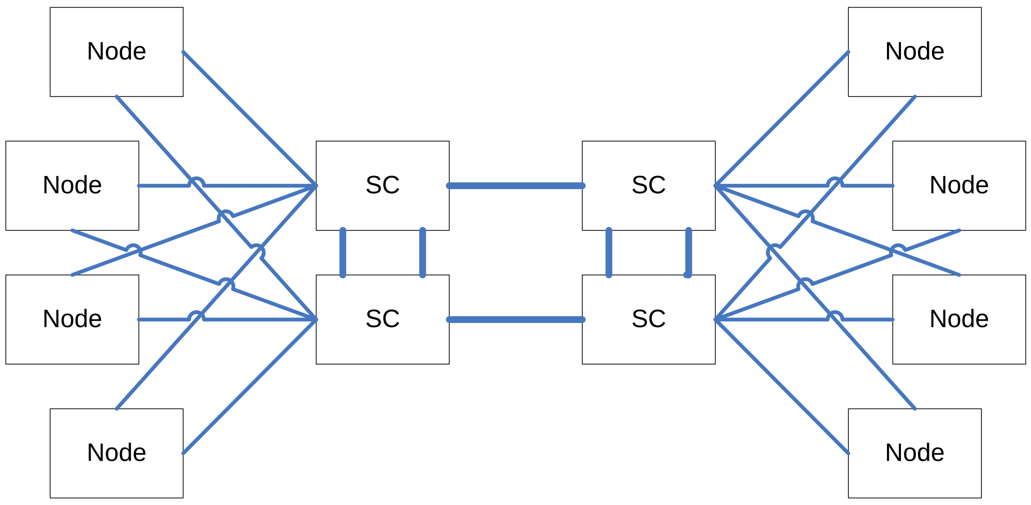


Figure 2.5: TTP/C multi-star topology

Messages from fault tolerant nodes are transmitted on both buses. In case of a message loss the message is not retransmitted. Each clock on the system cannot be perfectly synchronized. That is why the receiver resynchronizes by comparing the received time of a message and the expected received time.

2.7.1.2 Medium Access

The medium access is based on a Time Division Multiple Access (TDMA) scheme. The data transport between nodes is not point to point oriented but broadcast; this implies that every node receives all data transmissions available on the bus. Each node is allowed to transmit messages only in a predefined TDMA slot called SRU slot. Because the slots that the messages are transmitted are predefined, overload conditions cannot occur. The sequence of the SRU slots is called TDMA round. The TDMA rounds are equal but the length of the messages in the slots may differ. Several TDMA rounds executed after one another until the pattern is repeated are called a Cluster Cycle. The Cluster Cycle is repeated periodically throughout the execution of the system. In TTP/C there is no need for a message identifier because the messages can be identified by the point of time in which they transmit in the TDMA system. In the assigned slot a node sends frames on both channels, the frames on channel 1 and channel 2 may differ in their frame types, lengths and contents. The message overhead consists only from a 4bit header and 16bit CRC [TUW'97].

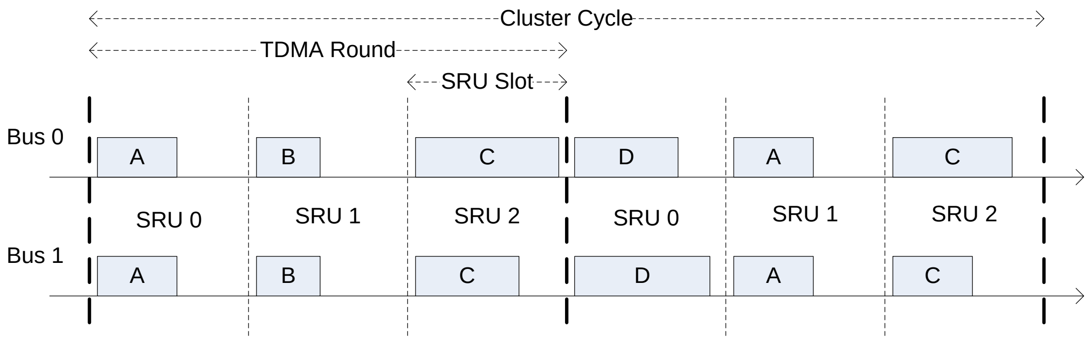


Figure 2.6: TTP/C medium access scheme

In figure 2.6 the slot allocation from the messages transmitted from the A, B, C, D and E nodes is shown. Nodes A and B are fault tolerant that’s why the messages are transmitted on both buses.

2.7.1.3 Message Description List

The slots used for data transmission are all predefined and the attributes of the messages sent and received are described in the Message Descriptor List (MEDL) which is saved at the Flash EEPROM. Also, the MEDL may contain node-local information and special setup data required for internal purposes of specific TTP/C controller implementations. All the nodes must have the complete MEDL downloaded in the controller and not only the node relevant information. The system will only work if all nodes agree on the order of the messages, the correct membership status of the nodes in the system, the integrity of the data, and the correct notion of global time.

2.7.1.4 Fault Tolerance

TTP/C supports three different ways to deal with faults that may occur on the nodes and network [Bannatyne'98].

- TTP/C supports active replication for fault tolerance nodes that are required to have guaranteed continuous operation. In the case that a fault occurs the node is replicated with an identical node.
- Another type of protection is the bus guardian. The bus guardian is an independent hardware device connected between the node and the bus. Only during the predefined transmission slot the bus guardian will allow the node to transmit to the bus. In this way, cases where the node starts broadcast continually a message (babbling-idiot) are avoided.
- To be able to manage faulty nodes the voting strategy is used. A fault tolerant unit consists of three independent nodes where the result of these three nodes are compared and analyzed. At the end there is only one verifiable result.

2.7.2 FlexRay

FlexRay is a deterministic and fault tolerant protocol that is targeted to be used in car applications. FlexRay is a result of a cooperation of BMW and Daimler Crysler for x-by-wire systems because of its fault-tolerant characteristics and redundant message transmission on two channels [Kopetz'01] [Consortium'04].

Important characteristics of FlexRay are the synchronous and asynchronous frame transfer, multi-master clock synchronization, guaranteed frame latency times and jitter during synchronous transfer, prioritization of frames during asynchronous transfer, error detection and signalling, error containment on the physical layer through an independent bus guardian device, and scalable fault tolerance [FlexRay'05, FlexRay'04].

2.7.2.1 Network Structure

The FlexRay supports data rates up to 10Mbit/s for each channel and can use two physically separated lines as a result giving a gross data rate of up to 20Mbit/s. FlexRay can be used in two different network topologies. The most common one is the classic bus topology which is in figure 2.7 and has = 1Mbit/s data rate per bus. The other kind of topology that can be used is the star topology represented in figure 2.8, where the nodes are connected point-to-point with the help of the active star couplers. This topology supports data rates above 1Mbit/s.

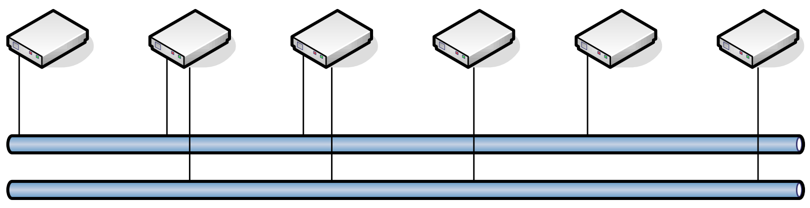


Figure 2.7: FlexRay bus topology

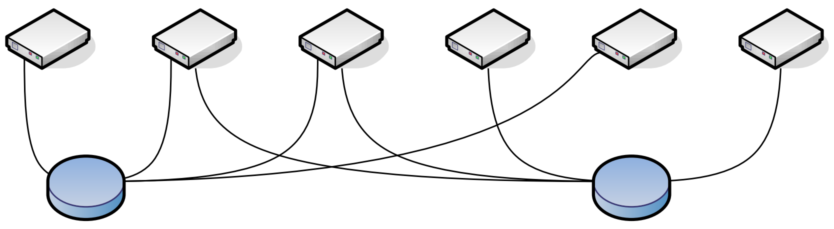


Figure 2.8: FlexRay star topology

2.7.2.2 Medium Access

To achieve maximum efficiency of the bandwidth a fault tolerant synchronized global time base is used. The nodes connected to the network use precise auto adjustable clocks which process with special algorithms the synchronization messages that are sent in the static segment of the transmission cycle as seen in figure 2.9. Because FlexRay supports synchronous and asynchronous data transmission, the transmission cycle is divided into the static and dynamic segments. FlexRay works according to the TDMA where the components have predefined time slots. In the static segment fixed time slots are allocated for predefined components to be transmitted. At the dynamic segment the slots are being created dynamically for maximum bandwidth efficiency and used by asynchronous data [FlexRay'04].

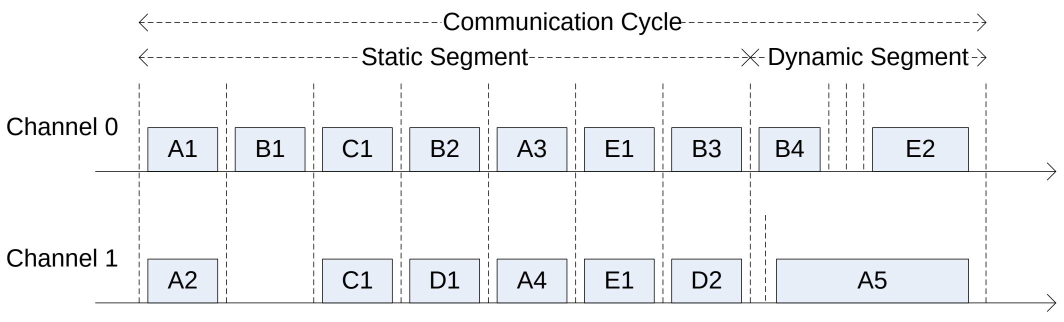


Figure 2.9: FlexRay medium access scheme

2.7.2.3 Fault Tolerance

To provide fault tolerant communication a FlexRay network node consists of a host processor, FlexRay controller, bus guardian and bus driver. As can be seen from figure 2.10 the bus driver connects the communication controller to the bus and monitors access to the bus. The bus guardian is instructed by the host processor the specific time slot that has to be used and allows the FlexRay controller transmit access only in these time slots. In addition the bus guardian can provide error detection feedback to the host processor to prevent further failure.

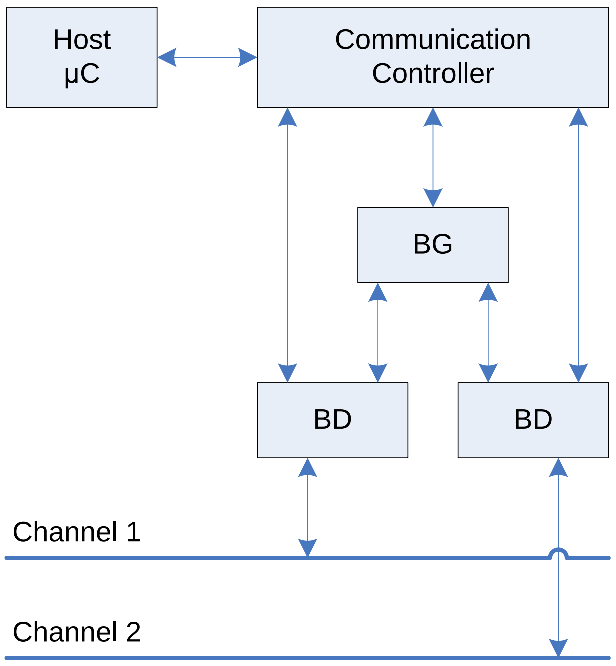


Figure 2.10: FlexRay Network node

2.8 Conclusion

This chapter starts with the background research of the thesis with an investigation of the fieldbuses, their history and characteristics. The structure of the fieldbus has been presented and the fieldbuses that have been adopted by the automotive industry. A more in depth analyses has been provided for TTP and FlexRay since both are modern communication protocols that can be characterised as fault tolerant, deterministic and high bandwidth.

Chapter 3 CAN and MilCAN protocol

3.1 Introduction

This chapter provides a brief insight in the Controller Area Network (CAN) and a more detailed analysis of the MilCAN protocol, where the later is an extension to the CAN protocol. MilCAN provides some advantages over CAN by offering a synchronised and deterministic data-link layer for time critical applications. The error detection capabilities of CAN are described, along with the technical aspects of the MilCAN specifications. Even though Controller Area Network provides a certain level of determinism and prioritization, it does not provide any scheduling to the operation of the network and neither higher layer control to the communication of the nodes. Because of these MilCAN is more suitable for military applications or other fully deterministic applications. MilCAN has been created by the MilCAN working group, to add determinism in the Transport layer of a CAN network without any hardware additions or modifications.

The MilCAN Working Group was established as a sub-group of the International High Speed Data Bus Users Group (IHSDB-UG) in 1999. Its target was to create a CAN based protocol that will be used for Military Land Vehicle subsystems communication. MilCAN provides deterministic and synchronised communication to the application layer and is located on top of the CAN network. MilCAN is capable to coexist on the same CAN bus with other CAN based protocols with no problems.

3.2 Controller Area Network

The Controller Area Network (CAN) is a high speed, serial communication protocol that was originally developed during the late 1980's for the automotive industry. Its main characteristics are high bit rate, high level of security, high immunity to electrical interface, low-cost multiplexed wiring and an ability to detect any errors produced. Due to these features the CAN serial communications bus has become widely used throughout the automotive, manufacturing, CAN powered devices include engine control units, sensors, anti-skid system, lamps, electric windows and others. Also there are other industry-standard protocols that are based on CAN, such as Allen-Bradley's DEVICENet, designed for the networking of PLCs and intelligent sensors [ODVA'04, Hitex'95].

The CAN communication protocol describes the method by which information is passed between devices, by broadcasting packets on the bus. Each packet defines a message frame that is sent by a node and received by the rest of the nodes connected to the segment. There are supported two types of message data frame formats by CAN, the Standard and Extended format. Different sizes of the Identifier field are the difference between the two, which is 11 bits for the Standard and 29 bits for the Extended. At the physical layer two complementary logical values are defined, dominant and recessive, but no logical values, like voltage levels, are specified. This adds flexibility into the specific implementation of the CAN bus, such as using electrical signal lines or optical fibre lines. The only requirement is that on a simultaneous transmission of a dominant and a recessive bit, the resulting bus value will be dominant.

The predominant CAN properties are:

- prioritisation of messages
- guarantee of latency times
- configuration flexibility
- multicast reception with time synchronisation
- system wide data consistency
- multimaster
- error detection and signalling
- automatic retransmission of corrupted messages as soon as the bus is idle again
- distinction between temporary errors and permanent failures of nodes
- autonomous switching off of defecting nodes

3.2.1 CAN Error detection

The error detection capabilities are based on the CAN controller's:

- Error Management Logic (EML)
- Receive Error Counter (REC)
- Transmit Error Counter (TEC)

To distinguish between temporary and permanent failures every CAN bus controller has two Error Counters, the REC and the TEC. The counters are incremented upon detected errors respectively are decremented upon correct transmissions or receptions. Depending on the counter values the state of the node is changed. The initial state of a CAN bus controller is Error active that means the controller can send active Error Flags. The controller gets in the Error passive state if there is an accumulation of errors. On CAN bus controller failure or an extreme accumulation of errors there is a state transition to Bus Off. The controller is disconnected from the bus by setting it in a state of high-resistance. The Bus Off state should only be left by a software reset. In figure 3.1 the three error state are shown, and the requirements for changing between them. After software reset the CAN bus controller has to wait for 128 x 11 recessive bits to transmit a frame. This is because other nodes may pend transmission requests. It is recommended not to start a hardware reset because the wait time rule will not be followed then [softing'05].

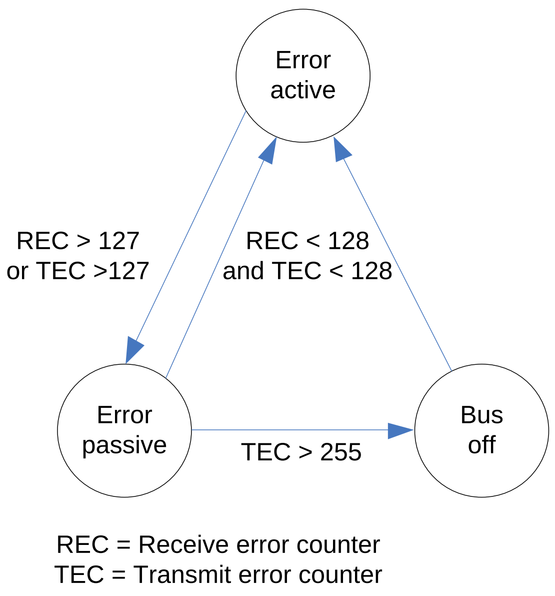


Figure 3.1: CAN error states

CAN controllers can be in one of the three states, depending on the error counter levels [ISO'93]:

Error active - An "error active" node can normally take part in bus communication and send an active error flag when an error has been detected. The active error flag consists of six dominant consecutive bits and violates the rule of bit stuffing and all fixed formats.

Error passive - An "error passive" node shall not send an active error flag. It takes part in bus communication, but when an error has been detected a passive error flag is sent. The passive error flag consists of six recessive consecutive bits. After transmission, an "error-passive" node will wait some additional time before initiating a further transmission.

Bus off - A node is in the state "bus off" when it is switched off from the bus due to a request of fault confinement entity in the "bus off" state, a node can neither send nor receive any frames. A node can leave the "bus off" state only upon a user request.

For detecting errors the MAC sub layer provides five mechanisms:

- Monitoring (transmitters compare the bit levels to be transmitted with the bit levels detected on the bus)

- 15-bit cyclic redundancy check
- Variable bit stuffing with a stuff width of 5
- Frame check
- Acknowledgement check

There are five different error types that can be detected in CAN. These error types are not mutually exclusive:

Bit error – When a node is transmitting a bit, it also monitors the bus. If the bit value sent is different from the one being monitored, then a bit error is detected at that bit time.

Stuff error – Since the frame field is coded by the method of bit stuffing, when there are six consecutive equal bit level a stuff error is detected at that bit time.

CRC error – The transmitter calculates the CRC sequence of the frame and is appended in the end of the frame. The receiver is also calculates the CRC in the same way as the transmitter did and compares the two. When they do not mach a CRC error is detected.

Form error – When a fixed-form bit field contains one or more illegal bits a form error is detected.

Acknowledgement error – The acknowledgement error is detected by the transmitter whenever it does not monitor a dominant bit during ACK slot.

3.3 MilCAN specifications

The MilCAN protocol is defined by three parts, the Physical Layer, the Data Link Layer and the Application Layer which are the same for both MilCAN A and B.

Physical Layer – The Physical Layer defines the physical connectivity between MilCAN devices and the topology of MilCAN networks. It includes the requirements for the segments bus, such as cable lengths and signal assignments and connector specifications. Also in the Physical Layer it defines the node transceiver characteristics and bit timing.

Data Link Layer – The Data Link Layer defines message types, frame format, node addressing of the network, message filtering and priority based bus access. The latest is one of the most important characteristics of MilCAN. Also here are defined the error detection capabilities and fault confinement.

Application Layer – The Application Layer is the highest layer where various MilCAN functionalities are defined such as message payload data byte order, message identifier assignment, deterministic transmission support, command distribution architecture and the various operational modes of MilCAN.

These three layers are described in a set of three working documents [MWG'03a] [Group'03] [MWG'03b] which now have been combined into one [MWG'09].

3.4 Physical layer

3.4.1 Physical Topology

There are two are two recommended physical topologies:

- Linear multi-drop
- Daisy chain

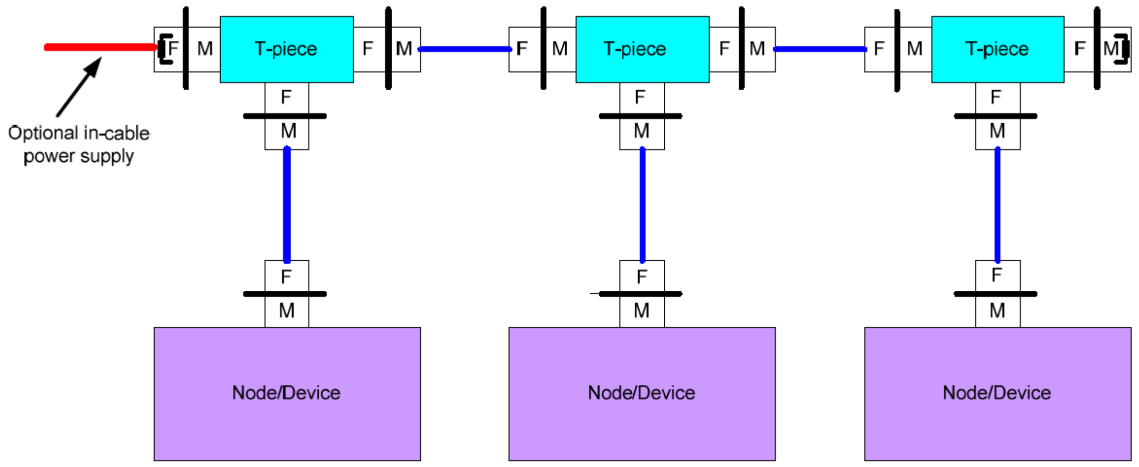


Figure 3.2: Linear multi-drop topology [MWG'09]

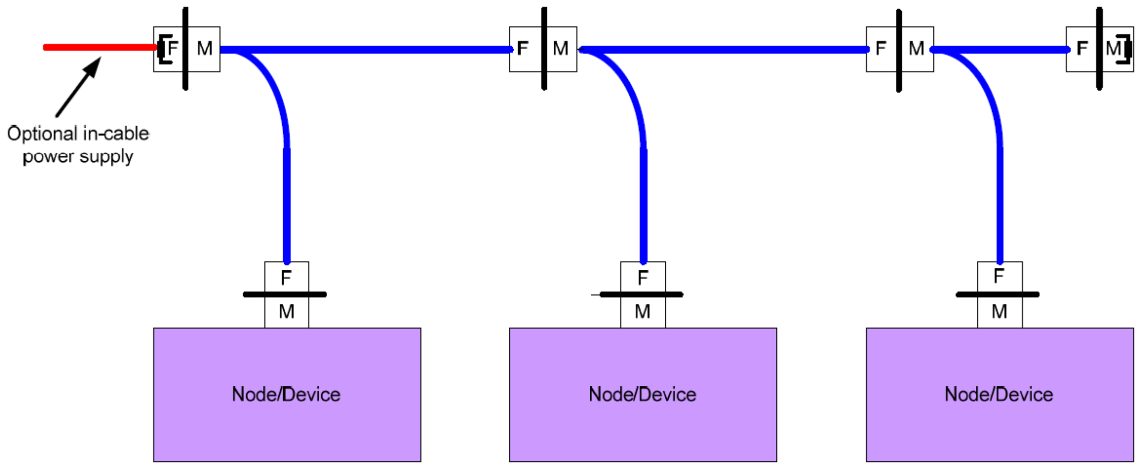


Figure 3.3: Linear multi-drop topology using bifurcated cables [MWG'09]

With the **Linear multi-drop** topology each device is linked to a bus with the use of a drop cable and a T-piece connector as can be seen in figure 3.2. To avoid the use of the T-piece connector another method can be used by replacing it with bifurcated cables as can be seen in figure 3.3. For both solutions the device is required to have only one bus connector.

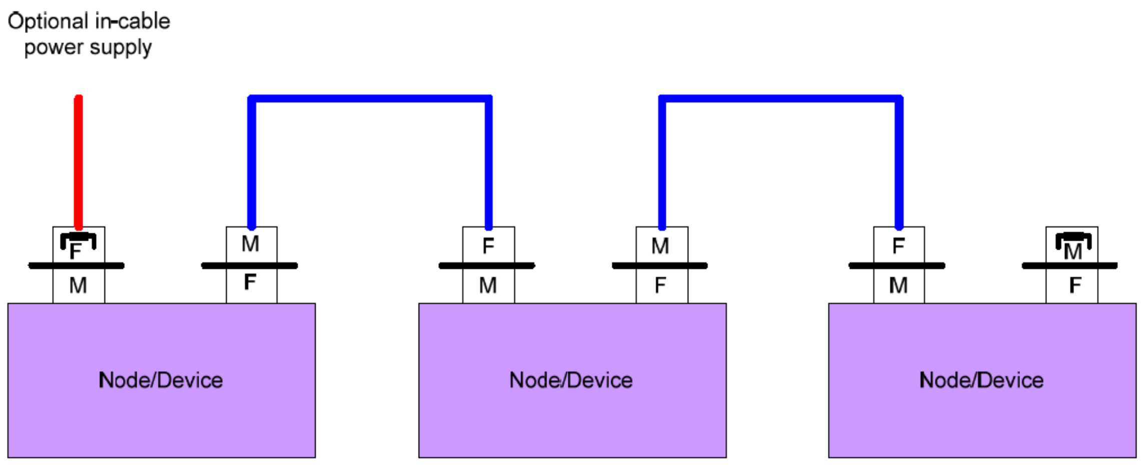


Figure 3.4: Daisy-chain topology [MWG'09]

With the **Daisy chain** topology the devices are connected to each other in series forming a chain. Each device has an input connector and an output connector. The output connector of one device is connected by cable to the input of another device; as a result this topology requires each of the devices to have two connectors as can be seen in figure 3.4. The two unused connectors at the two ends of the chain are terminated with a CAN bus terminator as instructed by the CAN protocol. Devices supporting the daisy-chain topology are capable of use in the multi-drop topology, utilising only the input bus connector. Equal cable length between devices is avoided to minimise standing waves. Similarly, drop cable lengths should not generally be equal. The optional implementation of in-cable power supply should fees into one end of the bus via a female connector, such that no male connectors carry live power or signal when exposed. [MWG'09]

3.4.2 Connector gender assignments

For the topologies shown in previous figures the recommended gender assignments are:

Cables	Male one end and female other end
Devices, multi-drop	Male
Devices, daisy-chain	Male input to female output
T-pieces, multi-drop	Male input to two female outputs

3.4.3 Maximum bus length and number of devices

Both the maximum bus length and the number of devices are specified in the ISO 11898 specification requirements.

3.4.4 Cable requirements

MilCAN specifications, group the bus cables according to their functionality. CAN Signal A group includes the lines required for the primary CAN bus. CAN Signal B group include the lines for the second CAN bus. It is recommended that the cables used must be shielded. It is up to the system designer to decide the specific details, depending the system requirements. It is recommended that there should be dedicated shields for CAN Signal and power pairs. When implemented the in-cable power is provided by the use of shielded twisted pair. The current rating of the cables is based on the systems requirements, whilst satisfying the power signal specification. Any lines designated as reserved are not be used by system designers, as they may be assigned a specification in future revisions.

The recommended connector for MilCAN systems is MIL-DTL-38999. A set of four connector configurations are suggested by the MilCAN specifications to allow a system developer to select a suitable connector for his application and thereby maintain compatibility by implementing the MilCAN configuration of that connector.

The configurations are:

MIL-DTL-38999-A Dual CAN Bus with In-Cable power

MIL-DTL-38999-B Dual CAN Bus without In-Cable power

MIL-DTL-38999-C Single CAN Bus with In-Cable power

MIL-DTL-38999-D Single CAN Bus without In-Cable power

The Bus termination is based on the ISO-11898 requirements with the following additions; Terminating resistors must be embedded inside a bus connector. Terminating resistors may be embedded into the network device only where a mechanism for switching them into and out of the network is also implemented either externally or internally.

The in-cable power supply is optional, depending the system design and it must comply with the MIL-Std-1275B. The output voltage is in the range of 18V to 32V and there is only a single input to the bus. Each device connected on the bus is required to have maximum current consumption of 500mA and be electrically isolated from all other external signals. If the above recommendations of individual shielding is used in the bus cables, these shields are suggested to be connected to the digital ground of the device, while the overall shield is connected to an earth connection (if exists) or the case.

3.4.5 Transceiver characteristics

CAN transceivers used by MilCAN devices and the resistance to electrical bus faults should conform to the physical medium attachment sub-layer as specified in ISO 11898.The CAN signal can be opto-isolated from the CAN controller and gets powered by isolated power supplies. Any additional propagation delays imposed by the opto-isolators are accounted and be chosen such that the maximum round trip interface delay time for a device comply with the bit timing requirements of MilCAN.

3.4.6 Bit timing

MilCAN devices can operate in one of these bit rates:

- 1Mbps
- 500Kbps
- 250Kbps

The selection mechanism for a device is described in the System Management Layer specification. MilCAN bit timing parameters are required to provide capability with other CAN based protocols by following the CANopen specifications and SAE J1939/11. The bit time oscillator tolerance is better than ± 0.1% in order to be allowed to operate on compliant MilCAN segment. For the 1Mbps the bit sample point shall be ≥ 75% of the bit time, preferably above 80%. For the 500Kbps and 250Kbps the bit sampling should be ≥ 87.5% of the bit time or later. The synchronisation jump width should be 1-time quanta, the sampling mode to single sampling and synchronisation to be “recessive to dominant” edges only. The round trip propagation time of a CAN interface, is less than 210ns at 1 Mbps and 300ns at 500Kbps and 250Kbps.

3.5 Data Link layer

3.5.1 Media Access Control

MilCAN provides a priority based bus access with an arbitration process that ensures the highest priority message will be transmitted first. When two nodes have a message to be transmitted, the one with the highest priority will be transmitted first. The process ensures no bus access conflicts occur that could result into loss of data and time. The transmitters of multiple nodes simultaneously transmit and receive one bit at a time, which they later compare to detect when another node is trying to transmit a message with higher priority.

When the message with higher priority has been sent and the bus is free, the arbitration procedure starts again for the following pending messages.

There are four types of frames that are used for message and status transfers. The **Data Frame** is the main type used for transferring data between nodes. The **Remote Frame** is used to trigger the transmission of the Data Frame from another node with the same identifier. The **Error Frame** is generated on bus errors. The **Overload Frame** is generated between successive Data Frames or Remote Frames to provide extra delay between their transmissions.

There are two types of Data Frames and Remote Frames, the **Standard** and **Extended**. The Standard has an 11 bits Identifier and the Extended has 29 bits (figure 3.5). When a frame is successfully transmitted or received an acknowledgement is accomplished during the transmission sequence of a frame. When no errors have occurred until the end of transmission of the frame’s EOF field, the transmitted message is considered valid. When an error occurs there is an automatic retransmission, which is not allowed by the MilCAN specification and should be disabled. Communication errors can optionally be reported and handled by a higher layer when detected by the CAN controller. All CAN controllers are required to keep error counters, one for transmit and one for receive. They are incremented and decremented based on a set of rules.

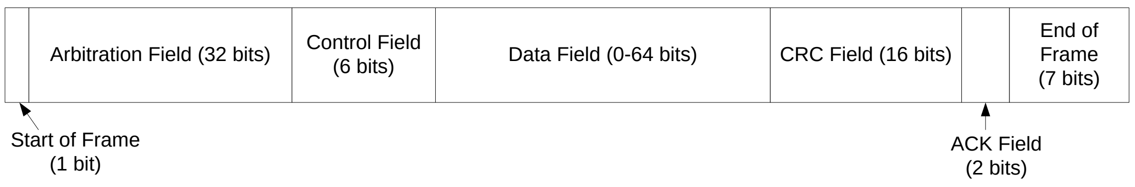


Figure 3.5: CAN frame format [MWG'09]

3.5.2 Logical Link Control

The 29-bit frame identifier is formatted as shown in figure 3.6. The **source address** of the node generated the frame is identified by the first eight bits (bits 0 to 7). This address is assigned to each node and it is unique. This allows the identification of the node that generated each message on the bus. The source address 0x00 is reserved and cannot be used by any node. The second and third bytes of the MilCAN frame identifier (bits 8-15 and 16-23) represent the secondary and primary type MilCAN **identifiers** respectively. Messages are grouped according their primary type based on their function and are also assigned a unique secondary type. The 24th bit is used to denote if the frame is either a **request** message (bit = 1) or a status/command message (bit = 0). Since the CAN bus can be shared by MilCAN and SAE J1939 devices, the **Protocol Type** (bit 25) is used to distinguish between the two protocols. For

MilCAN frames the bit is 1 and for the SAE J1939 frames is 0. Bits 26 to 28 store the **priority** of the MilCAN frame and are used when transmitting the frame. Based on the CAN media access control protocol, a frame has higher priority when the field is 0 and lowest priority when the field is 7. The priority assignment mechanism is defined in the Application Layer specification.

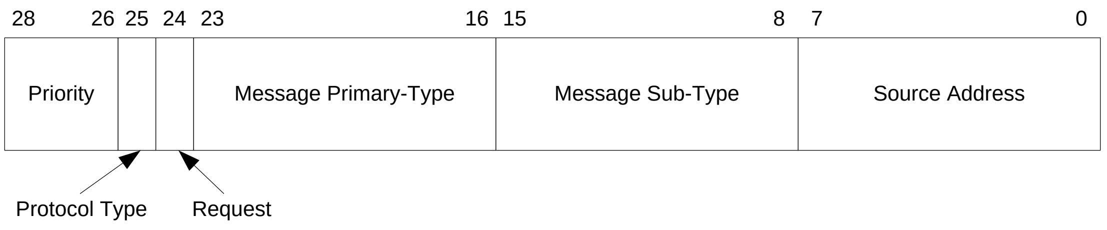


Figure 3.6: MilCAN frame identifier format [MWG'09]

When any errors occur on the CAN bus during operation, they may be reported to the application layer. The application then may execute the corrective procedures appropriately. No further actions are taken against any of these errors at the link layer. Where appropriate software is used to detect hardware errors of the network interface and then report them to the application layer. The application executes the appropriate procedure depending the error condition. No further actions are taken against any of these errors at the link layer.

The messages are classified only based on their primary-type and sub-type conveyed in the message identifier. The source address field is not used for functional distinction. For non-operational messages, a custom identifier that includes a destination address for the message can be used.

There are four operational message types:

- **Status/Command messages –** During normal operation these message types are the primary mode of communication. These type of messages are either periodic or event triggered.
- **Request messages** – Specific MilCAN messages can be requested from a remote device for a specific message primary-type and sub-type. The frame is distinguished from the Request bit (bit 24) of the identifier that is set to 1.
- **Non-operational messages** – Physically addressed messages use a fixed value of 0x31 as a primary-type. The sub-type identifier stores the physical address of the destination device. To broadcast the message to all the devices the 0x00 destination address is used.

- **Sync frame message** – The synchronous operation of MilCAN is controlled by the Sync-Frame message, to provide a coherent timing source to all the nodes. The generation of the Sync Frame is by the current elected Sync Master node. There is always a potential Sync Master per bus segment. When there are more than one potential Sync Masters, an election process is triggered. The election procedure is described in the Application Layer. The Sync Frame must have priority set to 0, the primary-type set to 0x00 and sub-type set to 0x80. The payload of the Sync Frame is a Sync Slot counter, which ranges between 0 and 1023. This counter is incrementing at each Sync Frame transmission and the generation frequency is system specific. Recommended values are 512Hz for 1Mbps CAN bus, 128Hz for 500Kbps and 64Hz for 250kbps CAN bus. The frequency change is described in the System Management specifications.

There are two message types related to the system configuration mode:

- **Enter configuration mode message** – The nodes on the bus can be requested to suspend operational mode and enter the configuration mode by using the Enter Configuration Mode message. During this mode the node can receive any required application specific reconfigurations. The message has a single byte data payload for the first, second and third message with payload the ASCII character 'C' , 'F' and 'G' respectively. The message has the highest priority '0', primary-type 0x00 and sub-type 0x81.
- **Exit configuration mode message** – This message shall be used to terminate the Configuration Mode. The message has a three byte data payload with value the three characters ASCII 'OPR'. The priority is '0', primary-type 0x00 and sub-type 0x82.

The **alive message** indicates the status of a node and it is required to be sent by all the nodes. The first payload of the payload when is '1' indicates a normal operation and when '0' faulty operation. The remaining 7 bytes of the data field may be used if required by the system design. The Alive message has a primary-type of 0x62 and the sub-type is used as a unique identifier to every node. The valid range for the node ID is 1 to 255. The system designer may use the source address to the node ID field. The transmission frequency of the alive message is 1 Hz.

For the generation and transmission of synchronous messages the Sync Frame Messages and the value of the sync counter need to be reported to the application layer. Nodes that are operating asynchronously are Sync Frame aware in order to transition between the system modes as defined in the Application Layer. Nodes that transmit asynchronous periodic messages, is their responsibility to control and maintain their triggering rate. Every periodic asynchronous message is required to have a priority of 2 or lower. The Sync Frame message and the Enter & Exit Configuration mode message are exempt which have a priority of 0.

Messages are transmitted depending their priority, the higher priority messages are transmitted before the lower priority messages. New messages with higher priority are queued before previously queued messages of lower priority. This requirement also applies to the CAN controller transmit buffers. This prevents “priority inversion” where lower priority messages are transmitted while higher priority ones are pending. When a MilCAN bus reaches a heavy busload or disruption, some queued messages could become invalid before being transmitted. To avoid this all the messages have a mortal attribute, which specifies if and when the message should be destroyed. When the mortal attribute is set to “TRUE” it is accompanied with a “time to live” value that defines the maximum time the specific message can be in a pending before it gets destroyed. When the mortal attribute is set to “FALSE” then the message is never be removed from the queue.

To reduce unnecessary load from the nodes CPU, filtering may be applied to the incoming messages that are not used by the node. The LLC layer of the CAN controller must not generate the “overload” frame of the ISO11898 CAN standard. CAN controllers selected for MilCAN use need to support disabling the “overload” frame. The LLC layer of the ISO11898 CAN standard supports the retransmission of frames that fail to be transmitted due to bus errors. This can result in uncontrollable continuous access to the bus, which could result to multiple transmissions of the same frame. In case of a disrupted CAN frame transmission, the application layer may execute the corrective procedures defined by the system design. Additionally the Remote Frame Request provided by the ISO11898 is not compatible with MilCAN’s logically addressed system and must not be used.

When the communication requires more than 8 bytes to be transferred, then the Multi-frame messages are used. The Multi-frame messages are a group of single frame messages that have the data required to be transmitted; fragmented in to multiple application payloads. Such frames can consist up to 251 individual single-frame messages. The transmission and reception of multiple Multi-frame messages at the same is allowed as long as they have different message identifier. Multi-frame messages are divided into three segments, the first, intermediate and last frame. The structure of a Multi-frame message can be seen in figure 3.7.

		Byte No						
		0	1	2	3	4	5	6
Frame No	0	Message Count	Byte Count			Reserved		
	1	Message Count	Data					
	↓	↓	Data					
	249	Message Count	Data					
	250	Message Count	CRC (Optional)		Reserved			

Figure 3.7: MilCAN Multi-frame structure [MWG'09]

Message Count – Represents the sequence number of the multi-frame message. The range of values that can have is from 0 to 250 to allow maximum of 251 frames per multi-frame message.

Byte Count – indicates the size of the multi-frame message and the range of values that it can get are 0 to 16449535 and the data is stored in Little Endian format.

CRC – is optional and provides an 8 or 16 bit CRC of the application data transferred by the Intermediate frames.

3.6 Application layer

3.6.1 Communication architecture

The payload bytes that are transmitted are in Intel format, where byte 0 is the first byte received and 7 the last byte. The bits within payloads bytes are transmitted with the most significant bit first and least significant bit last. Additionally MilCAN devices do not require sending an acknowledgement message. This will reduce unnecessary traffic on the bus. Depending the application and the system design, an appropriate mechanism is implemented to confirm the successful reception of the message.

The main objective of MilCAN is to provide deterministic communications to the devices that are connected to the network, while also to support non-deterministic communication. The Prioritised Bus Access with Bounded Throughput protocol will achieve the need for deterministic communication for the devices that require it and provides sufficient flexibility to accommodate those devices that do not.

The primary deterministic characteristics for MilCAN are:

- The message priority is assigned to each message depending the required delivery deadline of each message. Thus the message with the shortest delivery deadline is assigned the highest priority.
- The Sync frame is delivered by a Sync Master at a rate controlled by the shortest delivery deadline.
- There are multiple Sync Master devices and only one is elected as an active Sync Master which can output Sync Frames. To ensure there is only one active Sync Master, an election protocol is used which will elect a new master when the current Sync Frame Master fails.
- Every message has a minimum inter-arrival rate that is greater than or equal to the Primary Time Unit (PTU).
- When the messages and their inter-arrival times in a system are known, then the messages are pre-allocated to numbered slots during the design stage.
- The protocol accommodates three message priority categories, Hard Real Time (HRT), Soft Real Time (SRT) and Non Real Time (NRT).
- Since each node is responsible for its own message triggering and there will be some timing inaccuracies, the protocol allows these timing inaccuracies.
- In the event of failures, the protocol supports fault recovery.

MilCAN message frames are not generated more than once per PTU. That includes single-frame and multi-frame messages, where the application is responsible for this. Additionally depending the required delivery deadline of each message, different priority is assigned to them. The highest priority (lowest value) guarantee sorter delivery deadline. The available priorities are the following:

- **Priority 0 (HRT0)** – This priority messages gain immediate access to the bus. When a message from another node is transmitted, the HRT0 transmission is delayed till the transmission is complete. If two nodes have queued an HRT0 message, the one with the lowest CAN node identifier is transmitted first. The Sync Frame, EnterConfigMode and ExitConfigMode messages are the only messages defined by MilCAN to use HRT0 priority.

- **Priority 1 (HRT1)** – This priority messages have maximum latency of 1 PTU. All scheduled HRT1 messages of a network do not exceed the available bandwidth of 1 PTU from the one they were scheduled on.
- **Priority 2 (HRT2)** – This priority messages have maximum latency of 8 PTU. All scheduled HRT2 messages of a network do not exceed the available bandwidth of 8 PTU from the one they were scheduled on.
- **Priority 3 (HRT3)** – This priority message have maximum latency of 64 PTU. All scheduled HRT3 messages of a network do not exceed the available bandwidth of 64 PTU from the one they were scheduled on.
- **Priority 4 (SRT1)** – For this priority message the delivery latency is not guaranteed. Messages with this priority have high probability of being transmitted within 8 PTUs. These messages are transmitted on the available bus time not used by HRT0, HRT1, HRT2 and HRT3 messages, during 8 PTU period.
- **Priority 5 (SRT2)** – For this priority message the delivery latency is not guaranteed. Messages with this priority have high probability of being transmitted within 64 PTUs. These messages are transmitted on the available bus time not used by HRT0, HRT1, HRT2 and HRT3 messages, during 64 PTU period.
- **Priority 6 (SRT3)** – For this priority message the delivery latency is not guaranteed. Messages with this priority have high probability of being transmitted within 1024 PTUs. These messages are transmitted on the available bus time not used by HRT0, HRT1, HRT2 and HRT3 messages, during 1024 PTU period.
- **Priority 7 (NRT)** – Messages with this priority do not have any latency requirements.

Synchronous messages are triggered upon reception of the Sync Frame message that indicates the slot number of the MilCAN major cycle that each message had been scheduled on. The processing time of the Sync Frame on the device is taken into consideration, and the a fixed amount of time equal to the worst case response time is reserved in each PTU to ensure that all messages allocated to a particular slot are transmitted in that slot. The worst case scenario is calculated (figure 3.8) for the chosen hardware and software and ensure that this time is reserved in each slot when designing the message schedule.

An asynchronous event triggered message can be triggered in any slot. The required bandwidth is allocated at every slot to accommodate the possibly generated messages. Because this will waste bandwidth, event driven messages are reduced and even eliminated, and lower priority for event driven messages are used to minimise affecting delays. Devices with asynchronous messages (figure 3.9) although they do not need to associate the transmission of the asynchronous messages with the Sync Frame, they still need to receive and process it in order to detect operational mode changes of the network such as the initial transition from pre-operational to operational mode. When messages are transmitted periodically triggered from an internal timer, the period is equal or longer than one PTU. Based on the Data Link Layer asynchronous devices are restricted to the transmission of messages with priority 2 or lower.

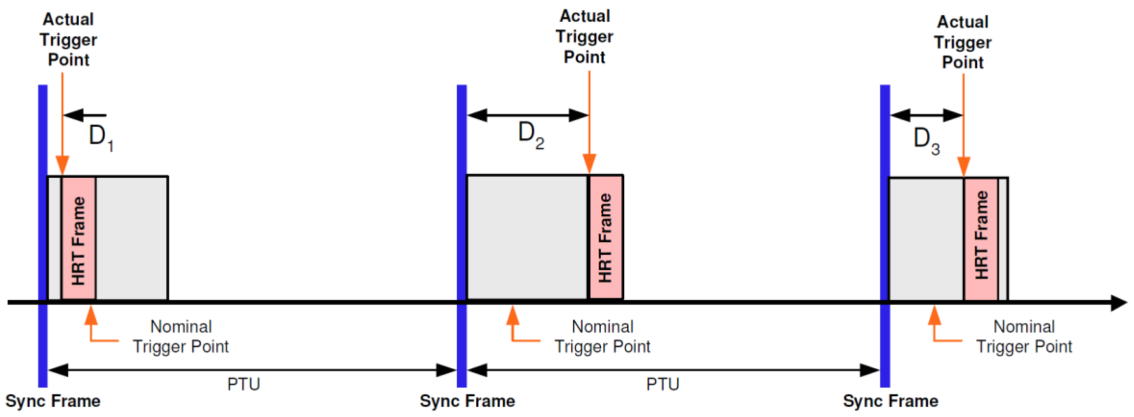


Figure 3.8: Response to Sync Frame [MWG'09]

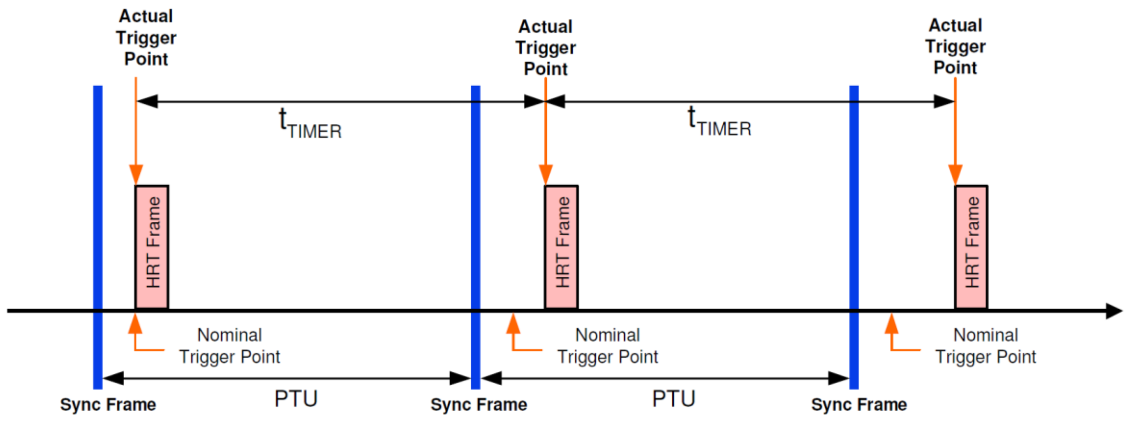


Figure 3.9: Asynchronous message triggering [MWG'09]

Military vetronic systems require deterministic message transfer to achieve predictable performance. This requires a synchronised operation between all the available devices and this is achieved within MilCAN by employing one of the network nodes as a Sync Frame message generator to provide this co-ordination. A single Sync Master makes the system vulnerable to its failure, a scheme of multiple nodes capable to assume Sync Master roles as a failover solution was established. Potential Sync Masters monitor the reception of the Sync Frame message, and when it is not receipt within a maximum timeout period a potential Sync Master takes over to transmit the Sync Frame message containing the next Sync Frame counter value.

The timeout period is greater than one PTU plus the time to transmit two messages of maximum length.

When a Potential Sync Master receives a Sync Frame message from a node with lower source address, it maintains the role of the Potential Sync Master. When a Potential Sync Master receives a Sync Frame message from a node with higher source address, it assumes the role of Sync Master and forcibly takeover by generating the next Sync Frame at a slightly smaller PTU. When a Sync Master receives a Sync Frame message from a node with lower source address, it assumes the role of Potential Sync Master and stops the transmission of the Sync Frame message. This procedure ensures that the node with the highest priority will eventually become the system Sync Master.

3.6.2 System modes

Three system modes are defined in the MilCAN protocol and are implemented in all devices. These modes are the Pre-Operational mode, Operational mode and System Configuration mode. The modes transitions are shown in figure 3.10.

- **Pre-operational mode -** Devices after power up, reset, loss of Sync Frames and upon exiting system configuration modes enter Pre-operational mode. During this mode only Sync Frame and Enter/Exit configuration messages are transmitted. Devices in Pre-operational mode will switch to Operational mode only when a Sync Frame is received, and when an enter system configuration mode message is received they switch to System Configuration mode.
- **Operational mode -** Devices enter Operational mode from Pre-operational mode upon reception of a valid Sync Frame. The devices exit Operational mode and enter Pre-operational mode following a reset or no Sync Frame message has been received within 8PTUs. The devices exit Operational mode and enter System Configuration mode upon reception of a valid enter system configuration message.
- **System configuration mode -** All devices enter System Configuration mode on request by a Configuration Master node. During this mode only system configuration mode messages are used, all operational messages, including Sync Frames, are suspended. The Enter System Configuration message is a sequence of three messages with same ID but with different payload for each message. The Enter System Configuration message is transmitted within 400ms. All devices connected to the bus, suspend Operational mode and enter System Configuration mode. The sequence of the message is received in the correct order. The enter System Configuration message is

transmitted every 1 second to notify devices that just came on-line after the segment has entered the System Configuration mode. The devices exit System Configuration mode and enter Pre-operational mode upon reception of the exit System Configuration mode message.

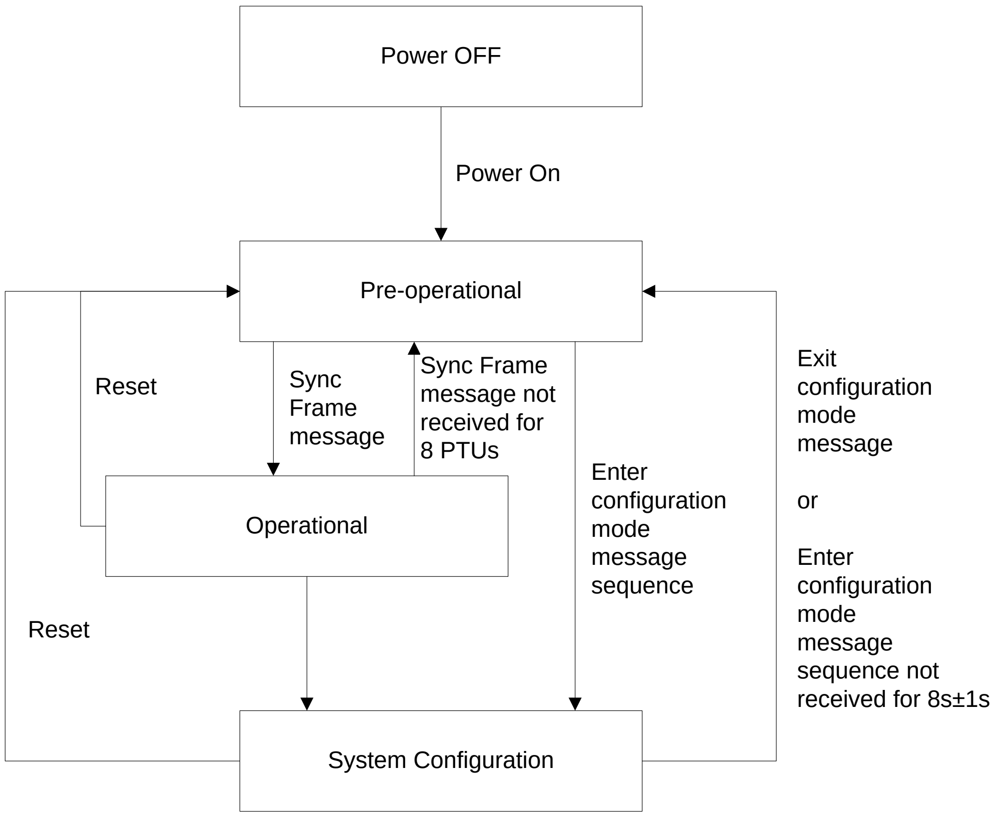

Power OFF
Power On
Pre-operational
Reset
Sync Frame message
Sync Frame message not received for 8 PTUs
Operational
Enter configuration mode message sequence
Exit configuration mode message
or
Enter configuration mode message sequence not received for 8s±1s
Reset
System Configuration

Figure 3.10: System modes [MWG'09]

3.6.3 Data distribution architecture

The communication between MilCAN devices is based on publisher/user basis. The nodes when they receive messages, will filter them based their message identifier and process only the ones required by the application layer.

3.6.4 Command distribution architecture

There are two forms of commands, the implicit and the system mode commands. The implicit commands indicate in the payload the change of the status of a parameter from a system function. The system mode commands indicate how the implicit commands are interpreted by the nodes, such as selection of data source when multi-instance addressing is used. Most of the command messages used in the system are implicit.

3.7 Conclusion

MilCAN is a protocol enhancement of the CAN protocol, which is developed by the military industry. Since MilCAN is a software based solution, is compatible with all already existing CAN enabled embedded microcontrollers and devices. By being compatible with all already existing CAN hardware is a flexible and affordable solution for real-time distributed systems. MilCAN is a deterministic, distributed real-time network which provides prioritised communication and direct synchronisation of all nodes. The frames that are transmitted on the bus by different nodes are synchronised and with the help of the arbitration mechanism of CAN the delivery latencies are guaranteed according to their priority. Since the network synchronisation is based on the MilCAN Master which in case of failure will be replaced by another potential master, the network operation resumes with minimal disruption.

Since the High Availability MilCAN is designed to operate on MilCAN nodes, it is very important its design to be based on the MilCAN protocol to assure capability between all the devices. The main characteristics of the High Availability MilCAN are based on the MilCAN internal components to provide the desired functionality.

Chapter 4 MilCAN Reconfiguration

4.1 Introduction

When there is the need to maintain a MilCAN node already installed in a vehicle, then the engineer has to spend time to access the node physically and may need to remove it. The above approach is time consuming and requires the knowledge and the tools to disassembly the node in order to gain access to it. Also during this operation the rest of the network is utilised unusable.

By using MilCAN Reconfiguration, many disadvantages of node maintenance are eliminated and through life capability management is provided. Node Reconfiguration covers the software side of the maintenance, like configuring the message set for every node and upgrading the firmware. With the Node Reconfiguration it's possible to connect to different vehicles and access the nodes remotely from a central location. In case a node has to be added or removed or just a software upgrade to many vehicles, by using a MilCAN Node Reconfiguration the software part of the modification could be completed automatically.

This chapter discuss the design of MilCAN Reconfiguration and explains the operation of its components. Each of the components are vital for the operation of the MilCAN Reconfiguration, which provides the functionality to remotely reconfigure and maintain any device connected on the MilCAN network without the need to reprogram the firmware on the nodes.

4.2 Design

Without the MilCAN Reconfiguration there was no ability to upload to a MilCAN node new firmware through the network. That makes it long and tedious operation for an engineer to develop and maintain an application. The MilCAN Reconfiguration although it can be used during operational mode, it is not advised since this will result in lost configuration messages and unexpected results. It should be used during configuration mode, where the operation of all the connected devices will be halted, and the network will be available only for the configuration commands.

The MilCAN Node Reconfiguration design consist three parts:

- Bootloader
- Communication protocol
- VSI GUI

Bootloader – The bootloader is located at the beginning of the firmware of the microcontroller, and is responsible for all the operations supported by the MilCAN Node Reconfiguration.

Communication protocol – The communication protocol allows the VSI GUI to communicate with the bootloader in the nodes.

VSI GUI – The VSI GUI is the front end that the user has to use to control the various operations the reconfiguration provides. The VSI GUI sends commands to the bootloader using the communication protocol.

To achieve this, the VSI GUI has to be able to send the firmware through the VSI Bridge to the nodes. Every node has the bootloader software at the beginning of the memory which is responsible to deal with receiving and saving the application firmware on the ROM memory of the node.

The bootloader is independent from the application firmware and allocated at the beginning of the ROM memory. For this reason the ROM memory is split in to two areas, the area that the bootloader is allocated and the area that application's firmware is allocated as seen in figure 4.1.

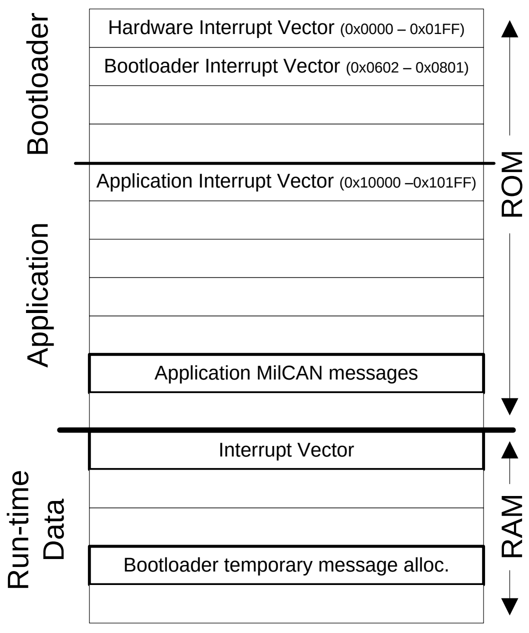

Bootloader
Hardware Interrupt Vector (0x0000 – 0x01FF)
Bootloader Interrupt Vector (0x0602 – 0x0801)
Application
Application Interrupt Vector (0x10000 –0x101FF)
Application MilCAN messages
ROM
Run-time Data
Interrupt Vector
Bootloader temporary message alloc.
RAM

Figure 4.1: Node software memory allocation

After a node reset the bootloader does a checksum verification to check the integrity of the application firmware, if it passes the check then the application firmware is executed by jumping the program counter at the beginning of the application firmware. From that point on the application is executed as normal.

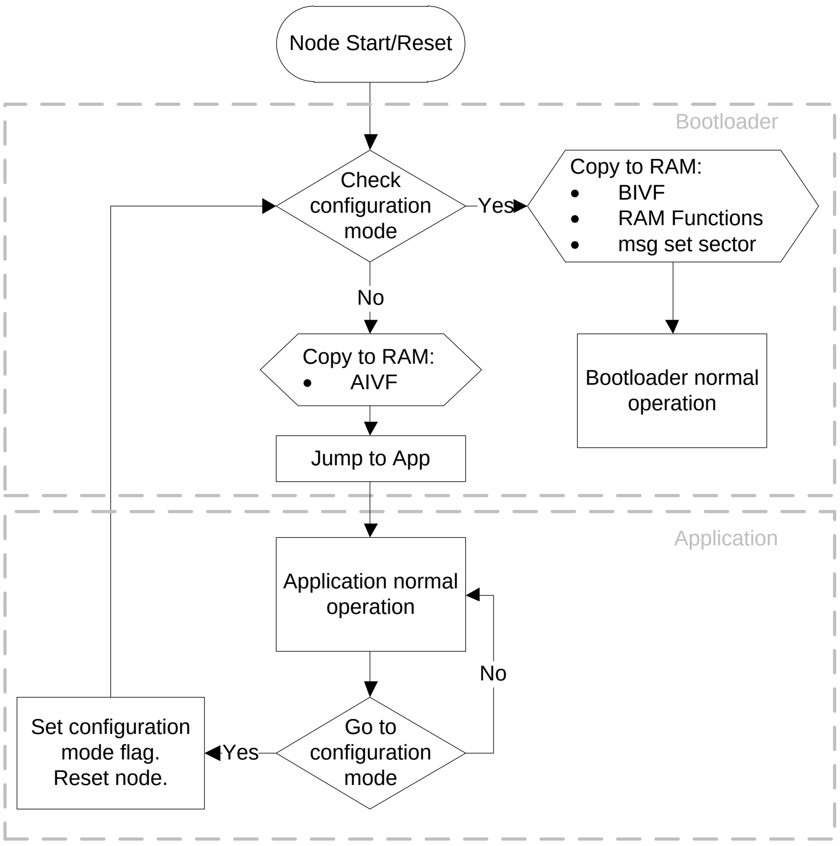


Figure 4.2: Flow of configuration mode

The configuration of the nodes has been adapted to include a unique identification number (serial number) for each device, decoupling the system from the use of the MilCAN Source Address to identify the individual nodes. This makes the development compliant with the MilCAN specification requirements. The feature that need to be supported are the following:

- Node software reset
- Node status checking (MilCAN mode, node mode)
- Configuration version checking
- Node ID and serial change
- MilCAN bus speed change
- Message set configuration

4.3 MilCAN Bootloader

4.3.1 Memory allocation

The microcontrollers that are used for the implementation are the phyCORE-167CS, where the ROM is allocated from 0x00000 to 0x3FFFF and RAM from 0x40000 and 0x7FFFF. Specific memory locations have been allocated for the operation of the bootloader and the applications which are shown at table 4.1.

Table 4.1: Node's memory map

Memory address	Description
0x00000-0x001FF	Hardware interrupt vector
0x00200-0x003FF	Bootloader interrupt vector forwarders
0x00400-0x005FF	Application interrupt vector forwarders
0x00602-0x00801	Bootloader interrupt vector
0x00802-0x009A9	RAM functions initial location
0x009AA-0x08000	Bootloader
0x10000-0x101FF	Application interrupt vector
0x101FF -0x2FFFF	Application
0x30000-0x30AFF	Application’s message set allocation
0x30B00-0x3FFFF	Application
0x40000-0x401FF	RAM interrupt vector
0x40200-0x403A8	RAM functions final location
0x403A9-0x6FFFF	RAM
0x70000-0x70AFF	Bootloader’s temporary message set allocation
0x70B00-0x7FFFF	RAM

Hardware interrupt vector – The hardware specific interrupts are pointing to this memory address.

Bootloader interrupt vector forwarders – Hardware interrupt forwarders used by the bootloader.

Application interrupt vector forwarders – Hardware interrupt forwarders used by the application.

Bootloader interrupt vector – Actual hardware interrupt vector for the bootloader.

RAM functions initial location – The location that the RAM functions are stored before are being copied to the RAM.

Bootloader – The location where the bootloader resides.

Application interrupt vector – Actual hardware interrupt vector for the application.

Application – The location where the application resides.

RAM interrupt vector – The location that the interrupt vector forwarders will be copied.

RAM functions final location – The location that the RAM functions will be copied.

RAM – This is unallocated RAM memory for general use.

Bootloader's temporary message set allocation – Is the location that the bootloader will copy the message set to edit it before it saves it back in ROM.

Additionally on these microcontrollers there is also an additional EEPROM connected through an I2C bus. On this memory the configuration of the node is saved. That includes the node ID, node serial and bus speed. This allows the bootloader to edit these settings without changing anything on the main ROM.

4.3.2 Dynamic interrupt vector

The nodes that are MilCAN Reconfiguration enabled have both a bootloader and an application. To ensure normal operation of the bootloader or the application the interrupt calls should be forwarded appropriately. The bootloader and the application require different interrupt vectors which are going to be allocated by the compiler when configured appropriately. To be able to have a dynamic interrupt vector and the interrupt calls to be forwarded to the bootloader or the application during runtime, the RAM is used as a dynamic forwarder as seen in figure 4.3.

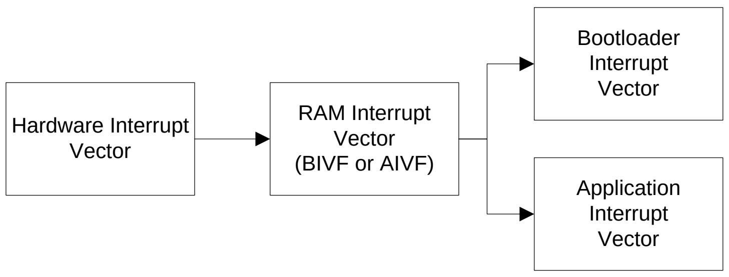

Hardware Interrupt Vector
RAM Interrupt Vector (BIVF or AIVF)
Bootloader Interrupt Vector
Application Interrupt Vector

Figure 4.3: Dynamic interrupt vector

To achieve that, custom interrupt vectors forwarders were created with the help of assembly programming and the use of the **JMPS** instruction which what it does is an unconditional jump to any target [ARM'04]. These three interrupt vectors are the Hardware Interrupt Vector, Bootloader Interrupt Vector Forwarders (BIVF) and the Application Interrupt Vector Forwarders (AIVF). When the device is restarted the bootloader copies to the RAM the appropriate interrupt vector forwarders. When the bootloader needs to be executed then the BIVF is copied to the RAM and if the application needs to be executed then the AIVF needs to be copied to the RAM. When an interrupt occurs then it jumps from the Hardware Interrupt

Vector to the RAM Interrupt Vector from where again it is forwarded by the BIVF or AIVF as can be seen in figure 4.3. All the interrupts are dynamically forwarded except the reset interrupt which is always forwarded to the bootloader's interrupt vector, in order to execute the bootloader every time the node is reset.

4.3.2.1 Hardware interrupt vector

The Hardware Interrupt Vector is located at the beginning of the ROM and is allocated in the memory address range 0x00000 to 0x001FF. Each memory address in this range represents an interrupt. At this location a custom interrupt vector is created that redirects any call to the corresponding address in the RAM interrupt vector with the use of the VECTAB Linker Directive which allows you to specify a starting address (offset) for the interrupt vector table. By default, the starting address is 0000h [ARM'05]. The RAM interrupt vector is located in 0x40000 to 0x401FF memory address.

4.3.2.2 Bootloader Interrupt Vector Forwarders (BIVF)

The BIVF is located in the memory address range 0x00200 to 0x003FF and it is pointing to the Bootloader's Interrupt Vector located in memory address range 0x00602 to 0x00801. Depending the device operation, this interrupt is copied in the pre-allocated location on RAM which has address range of 0x40000 to 0x401FF.

4.3.2.3 Application Interrupt Vector Forwarders (AIVF)

The AIVF is located in the memory address range 0x00400 to 0x005FF which is pointing to the Application's Interrupt Vector located in memory address range 0x10000-0x101FF. As the BIVF, the AIVF will be copied in the pre-allocated RAM area in order to forward any calls to the Application Interrupt Vector located in 0x10000 to 0x101FF.

4.3.3 Message configuration

When a node needs to be reconfigured without flashing a new firmware the VSI GUI can with the appropriate commands reconfigure the message list with the help of the bootloader. The bootloader is responsible to make the correct changes permanently on the applications firmware. To achieve that, the application's message list must be located in to a predefined location on the ROM in order for the bootloader to know the exact location that need to be modified. That location is in the memory range of 30000h to 30AFFh for the specific device that is used. Since the hardware ROM does not allow to edit specific location in the memory, the whole sector needs to be deleted before it is rewritten, the bootloader loads that sector data in to the RAM where it edits it and applies any changes that have been received by the user through the VSI GUI. The sector data include both the message set and part of the applications data. When the whole configuration is complete and the user decides to save the changes, then the bootloader deletes the sector that the message list is located in the ROM which has memory range of 0x30000 to 0x40000. When the ROM sector is deleted successfully, it copies the data stored in the RAM to the ROM. After the completion of the operation, the bootloader will verify the operation by comparing the written data to ROM with the ones stored in RAM. If the verification is successful then it will wait for further commands otherwise it will try to rewrite the data. The whole process can be seen in figure 4.4.

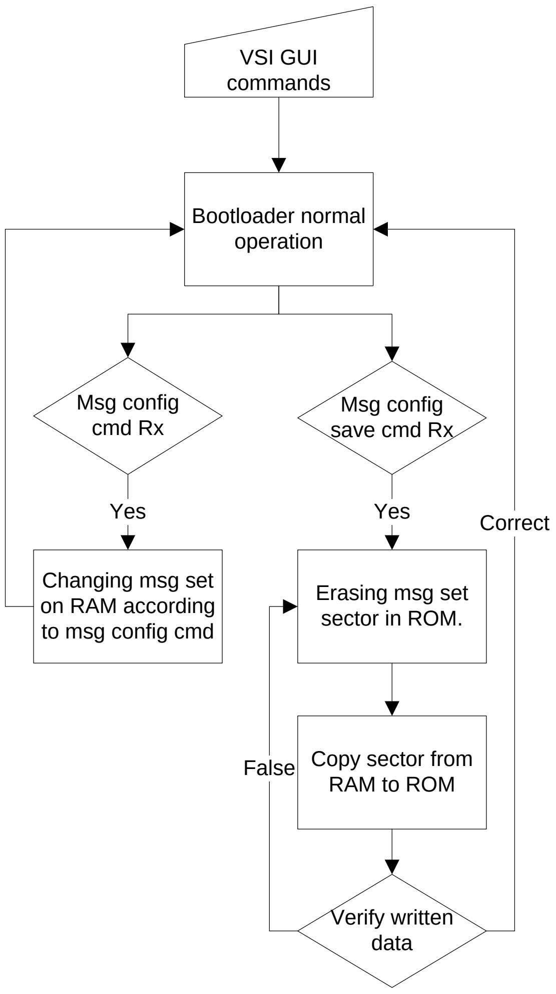

VSI GUI commands
Bootloader normal operation
Msg config cmd Rx
Msg config save cmd Rx
Yes
Yes
Correct
Changing msg set on RAM according to msg config cmd
Erasing msg set sector in ROM.
False
Copy sector from RAM to ROM
Verify written data

Figure 4.4: Bootloader's message configuration flow

4.3.4 RAM Functions

The above operations have a major restriction brought by the device. When any ROM operations are carried such as deleting or writing the operation must be executed from the RAM. It is not possible to write or delete to the ROM with the instructions being located in the same ROM. For these operations, a specific set of instructions that manage the ROM are generated. These instructions are named RAM Functions and located in a predefined location in the memory address 0x00802 to 0x009A9 in the ROM. These instructions although are located to the memory location mentioned above, are not able to be executed from there. The reason is that the compiler is configured to reallocate them as if they are located in the memory address range 0x40200 to 0x403A8 which is in the RAM. When the device goes in to configuration mode, the bootloader copies the RAM Functions in to the appropriate locations in to the RAM which from there are going to be able to be executed. This way when there is a need to edit a sector in ROM the RAM functions will take care of the operation without the execution to be located in ROM. When the RAM Functions are executed the rest operations of the device are coming to a halt.

The functions located in the RAM are:

- Data polling (compares the data from ROM to RAM)
- Erase chip (deletes the content of the whole chip)
- Erase sector (deletes a specific sector)
- Program flash (copies data to ROM)

4.3.5 Program flash memory

In order to save the application firmware on the node it has to be saved on the ROM memory. There is one difficulty to do this; the flash memory cannot be program with code located in the same flash memory. This means that the flash programming algorithms must be copied to and executed in the internal RAM. To manage this, the functions that are used to write on the ROM are copied on the RAM just after the interrupt vector that is placed on the RAM (ram_functions). Specific settings on the compiler allow code to be executed from the RAM. These settings, compile the ROM programming functions as if they were placed on the Ram in the first place.

4.4 MilCAN Reconfiguration Protocol

A specific protocol has being designed for the Node Reconfiguration operations in order to utilize all the functionalities that the bootloader can provide. It is a new protocol that insures proper MilCAN Reconfiguration operation. It provides a wide range of operations, where each of them is verified with the use of acknowledgment messages. The whole protocol can be found in the appendix. The operations that the bootloader provides are:

- Enter programming mode
- Software reset
- Status check
- Version check

- Node ID change
- Node speed change
- Message configuration
- Erasing app sectors (erase the sectors that are allocated for the application)
- Erasing chip (erase the whole ROM chip including the bootloader)
- Address set
- Programming data
- Checksum (after completing programming the checksum for the application)

4.4.1 Enter programming mode

When the application is operating normally the **Enter programming mode** command will leave the application and execute the bootloader for the node to become ready for configuration. The application of the node has to recognise this command. The structure of the command can be seen in table 4.2. When the node receives this command before the switch, it sends back an acknowledge message with message ID 0xBE25.

Table 4.2: Configuration protocol Enter programming mode

Header	Payload							
Msg ID	0	1	2	3	4	5	6	7
0xBE24	Slave ID							

4.4.2 Software reset

When the **Software reset command** is received during normal node operation the node will be restarted. Before the restart of the node it will transmit back an acknowledge message with message ID 0xBE2B and the same payload. The structure of the command can be seen in table 4.3.

Table 4.3: Configuration protocol Software reset

Header	Payload							
Msg ID	0	1	2	3	4	5	6	7
0xBE2A	Slave ID							

4.4.3 Status & Version check

The **Status check** command can be used for checking the operation mode that MilCAN is in and if the node is in the bootloader or the application. The status request should have message ID 0xBE2E and Payload 0 the slave ID that the status is requested from. The structure of the result message that will be sent from the node can be seen in table 4.4.

Table 4.4: Configuration protocol Status check (result)

Header	Payload							
Msg ID	0	1	2	3	4	5	6	7
0xBE2F	Slave ID	MilCAN mode	Node mode					

The **Version check** command checks the version of the application in the node. The request should have message ID 0xBE30 and Payload 0 the slave ID that the status is requested from. The structure of the result message that will be sent from the node can be seen in table 4.5.

Table 4.5: Configuration protocol Version check (result)

Header	Payload							
Msg ID	0	1	2	3	4	5	6	7
0xBE31	Slave ID	Version	Version	Version	Version	Version	Version	Version

4.4.4 Node ID & speed change

Every MilCAN node has a unique ID. This ID can be changed with the **Node ID change** command. For this operation the unique serial that every node has, will be used. In table 4.6 the structure of this command is shown. To acknowledge the command the node retransmits the same message but with message ID 0xBE33.

Table 4.6: Configuration protocol Node ID change

Header	Payload							
Msg ID	0	1	2	3	4	5	6	7
0xBE32	Slave ID	MILCAN ID1	MILCAN ID2	NODE SERIAL	NODE SERIAL	NODE SERIAL	NODE SERIAL	NODE SERIAL

The MilCAN operational speed can be changed with the **Node speed change** command. The structure of the command can be seen in table 4.7. After the reception of the command the node need to acknowledge the reception of the command by sending the same message with message ID 0xBE35.

Table 4.7: Configuration protocol Node speed change

Header	Payload							
Msg ID	0	1	2	3	4	5	6	7
0xBE34	Slave ID	NODE SPEED A_1	NODE SPEED A_1	NODE SPEED A_1	NODE SPEED A_1			

4.4.5 Message configuration

To be able to configure the list of messages on the node the **Message configuration** need to be used. By using this command the priority, message ID, start frame and frequency of a MilCAN message can be changed. The structure of the command can be seen in table 4.8. After the reception of the command the node acknowledge with the transmission of the same payload but with message ID 0xBE51.

Table 4.8: Configuration protocol Message configuration

Header	Payload							
Msg ID	0	1	2	3	4	5	6	7
0xBE50	Slave ID	Primary ID	Secondary ID	New Priority	New Primary ID	New Secondary ID	New Start Frame	New Cycle

After the completion of the message configuration in order to save the configuration the **Message configuration save** command is used, which can be seen in table 4.9. As an acknowledgement the node sends the same message but with message ID 0xBE53.

Table 4.9: Configuration protocol Message configuration save

Header	Payload							
Msg ID	0	1	2	3	4	5	6	7
0xBE52	Slave ID							

4.4.6 Flashing application firmware

Before the actual firmware is transferred to the node the flash memory needs to be prepared. To do so the **Erase application sector** needs to be used. During this command only the application section of the flash memory is erased. The structure of the command can be seen in table 4.10. After reception the node sends an acknowledgement with message ID 0xBE27 and the same payload. When the command is completed, the flash is ready to be reprogrammed with a new firmware remotely.

Table 4.10: Configuration protocol Erase application sector

Header	Payload							
Msg ID	0	1	2	3	4	5	6	7
0xBE526	Slave ID							

To load a new firmware to the nodes through the network the following commands need to be used. The binary file that has been generated from the compiler is based to the Intel HEX file format [Wikipedia'10a].

1. **Start code**, one character, an ASCII colon ':'.
2. **Byte count**, two hex digits, one of byte (hex digit pairs) in the data field
3. **Address**, four hex digits, a 16-bit address of the beginning of the memory position for the data. Limited to 64 kilobytes, the limit is worked around by specifying higher bits via additional record types. This address is big-endian.
4. **Record type**, two hex digits, 00 to 05, defining the type of the data field.
5. **Data**, a sequence of n bytes of the data themselves, represented by 2n hex digits.
6. **Checksum**, two hex digits - the least significant byte of the two's complement of the sum of the values of all fields except fields 1 and 6 (Start code ":" byte and two hex digits of the Checksum).

To follow the same format for the communication protocol the **Address set** command is send first from the server to the node (table 4.11). The node has to reply to the server with an acknowledge message which has the same payload and message ID 0xBE21. After the address is set, the server will transmit the data with the **Programming data** command (table 4.12). Again the node is responsible to acknowledge the reception of the command by transmitting back to the server a message with the same payload and message ID 0xBE23.

Table 4.11: Configuration protocol Address set

Header	Payload							
Msg ID	0	1	2	3	4	5	6	7
0xBE20	Slave ID	Sequence Number	Data Size	Address Upper Bytes	Address Lower Bytes	Checksum		

Table 4.12: Configuration protocol Programming data

Header	Payload							
Msg ID	0	1	2	3	4	5	6	7
0xBE22	Slave ID	Sequence Number	Data Upper Bytes	Data	Data	Data	Data	Data Lower Bytes

4.5 VSI GUI

To utilise the network related data, a user interface has been developed. The original VSI GUI has the ability to deal with high-level operations of a VSI network (controlling VSI Bridge) as well as the sub-system level; transmitting and receiving MilCAN frames from selected segments. The purpose of the VSI GUI is to provide a user interface for viewing the VSI network from a higher level of abstraction, and to provide tools for MilCAN specific investigation.

The VSI GUI has the ability to store a copy of a live database, to be used as a project file when the GUI is not connected to the network. Since the database can have detailed information of the network, the user is able to store many different configurations on an off-line basis for all the components of the VSI network, including bridge configurations, message flooding, statistics and node firmware [Valsamakis'06].

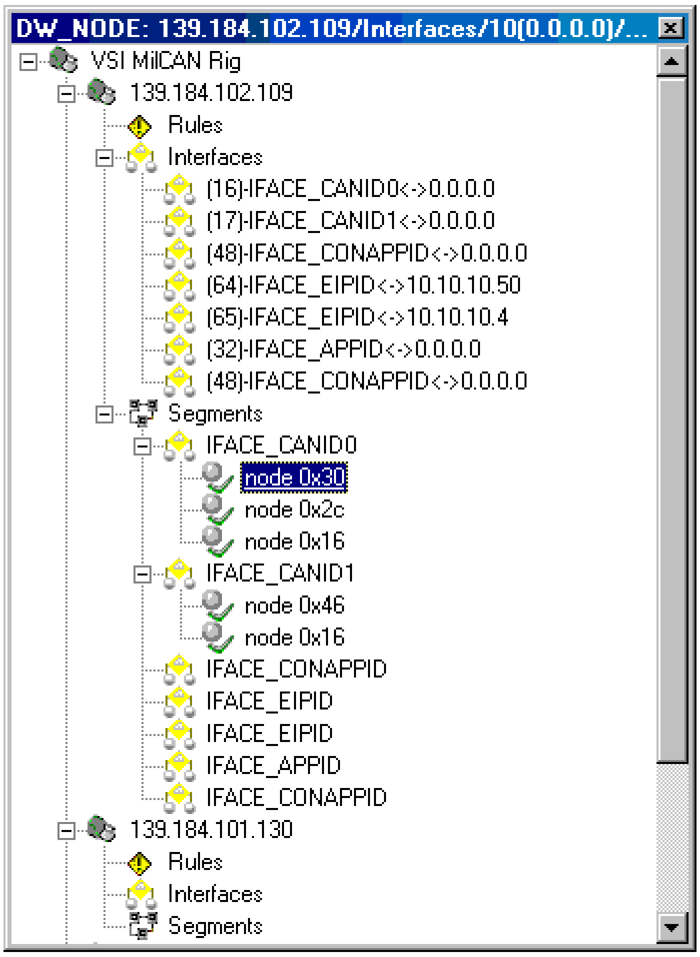
Specifically, the following features and services are provided by the VSI GUI:

- On/Off line topology view of interconnected VSI Bridges and their adjacent segments.
- Saving/Loading different VSI system databases.
- Controlling VSI bridge, segment and node status.
- Retrieving firmware information from MilCAN nodes anywhere in the network
- Retrieving MilCAN flooding from network
- Reconfiguring VSI NEC MilCAN nodes with new firmware while connected.
- Simulating message generation of a MilCAN cycle for individual bridges, segments and nodes.
- Reconfiguring message parameters for individual nodes.

Following the VSI standard guidelines, the preferred programming language used was C++. The VSI GUI is inheriting its implementation from the VSI-System COM/ATL library, which supports VSI GUI as a client application, in the same way as the VSI GUI has been supported. To take advantage of modern workstation desktop capabilities, this GUI supports docking windows, allowing the user interface to be setup as user prefers, and to support multiple screen displays.

4.5.1 Network View

The VSI GUI is capable of analysing the Flooding and VSI bridge tables of the VSI storage, to create a tree structure of the connected network. Figure 4.5 shows an example layout of a VSI network. The items are populated according to the VSI object hierarchy up to sub-system level (Vehicle, interconnected VSI Bridges and their current configuration), and then on system-level, with node objects existence and position is derived from the flooding information.



```
DW_NODE: 139.184.102.109/Interfaces/10(0.0.0.0)/...
VSI MilCAN Rig
    139.184.102.109
        Rules
        Interfaces
            [16]-IFACE_CANID0<->0.0.0.0
            [17]-IFACE_CANID1<->0.0.0.0
            [48]-IFACE_CONAPPID<->0.0.0.0
            [64]-IFACE_EIPID<->10.10.10.50
            [65]-IFACE_EIPID<->10.10.10.4
            [32]-IFACE_APPID<->0.0.0.0
            [48]-IFACE_CONAPPID<->0.0.0.0
        Segments
            IFACE_CANID0
                node 0x30
                node 0x2c
                node 0x16
            IFACE_CANID1
                node 0x46
                node 0x16
            IFACE_CONAPPID
            IFACE_EIPID
            IFACE_EIPID
            IFACE_APPID
            IFACE_CONAPPID
    139.184.101.130
        Rules
        Interfaces
        Segments
```

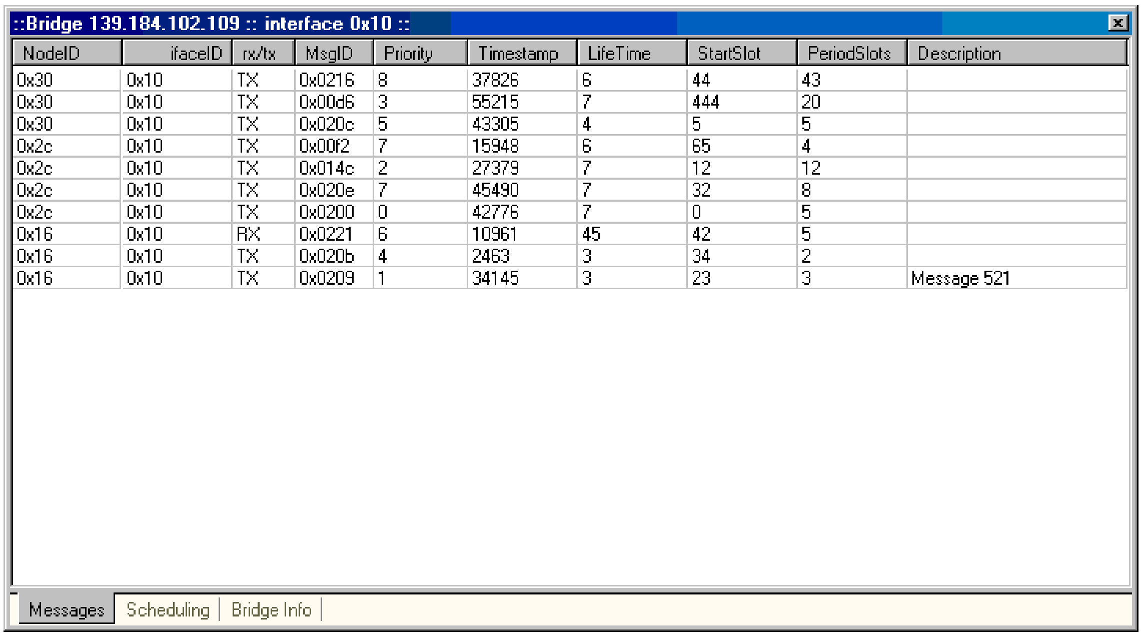
Figure 4.5: VSI Network Topology GUI View

When selecting an object such as a segment or node, the flooding information grid view of the specific object is displayed in the GUI. The flooding information grid view displays the parameters shown in table 4.13.

Table 4.13: VSI GUI flooding information grid view parameters

iFaceID	The adjacent interface – segment where the message is generated or received.
nodeID	The node identifier of the node where the messages is generated or received.
RX/TX ID bit	Whether a message is transmitted or received by the specific node.
MsgID	The unique identifier of the message.
Priority	Message MilCAN priority.
StartSlot	MilCAN cycle slot when periodic message begins.
PeriodSlot	Period in MilCAN message transmission cycle slots.
Description	Text description for this message.

The VSI GUI flooding information grid view is shown in figure 4.6. The displaying grid is editable by the user, where the fields MsgID, Priority, StartSlot, PeriodSlots and message description can be changed and saved.


::Bridge 139.184.102.109 :: interface 0x10 ::

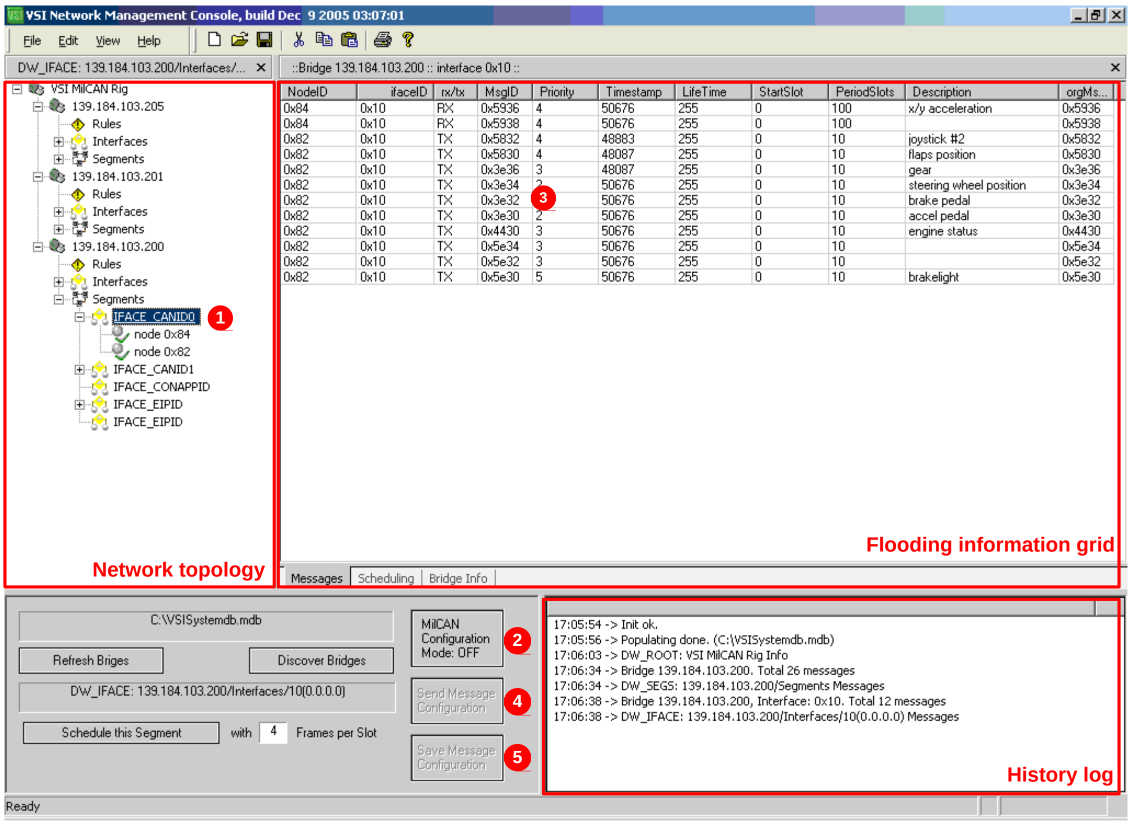
NodeID	ifaceID	rx/tx	MsgID	Priority	Timestamp	LifeTime	StartSlot	PeriodSlots	Description
0x30	0x10	TX	0x0216	8	37826	6	44	43	
0x30	0x10	TX	0x00d6	3	55215	7	444	20	
0x30	0x10	TX	0x020c	5	43305	4	5	5	
0x2c	0x10	TX	0x00f2	7	15948	6	65	4	
0x2c	0x10	TX	0x014c	2	27379	7	12	12	
0x2c	0x10	TX	0x020e	7	45490	7	32	8	
0x2c	0x10	TX	0x0200	0	42776	7	0	5	
0x16	0x10	RX	0x0221	6	10961	45	42	5	
0x16	0x10	TX	0x020b	4	2463	3	34	2	
0x16	0x10	TX	0x0209	1	34145	3	23	3	Message 521

Figure 4.6: VSI GUI flooding information grid view

4.5.2 Message configuration

To manage an uninterrupted reconfiguration of a node the following routine needs to be followed. The VSI GUI needs to establish connection to the segment that the desired node is connected and retrieve the network topology as displayed in the left part of figure 4.7. After successfully retrieving the topology the following steps need to be followed as shown in figure:

1. The desired CAN interface of the node that will be reconfigured needs to be selected.
2. Put the node in configuration mode by loading the bootloader by pressing the **MilCAN Configuration Mode** button. (The button will change and inform the user that the node is configuration mode after it receives the acknowledgement message. The acknowledgement will show also in the **History log**.)
3. Edit the desired message with the new settings by clicking on the existing value and type a new one. (Only the fields MsgID, Priority, StartSlot, PeriodSlots and message description are editable.)
4. Press the **Send Message Configuration** button to send the configuration to the node. Immediately after that the acknowledgement message will be displayed in the **History log**. (Steps 3 and 4 can be repeated if more messages need to be configured.)
5. Press the **Save Message Configuration** button in order to permanently save the messages in the ROM.


VSI Network Management Console, build Dec 9 2005 03:07:01
File
Edit
View
Help
DW_IFACE: 139.184.103.200/Interfaces/...
::Bridge 139.184.103.200 :: interface 0x10 ::
1
2
3
4
5
Network topology
Flooding information grid
History log
Messages
Scheduling
Bridge Info
Refresh Briges
Discover Bridges
Schedule this Segment
MilCAN Configuration Mode: OFF
Send Message Configuration
Save Message Configuration
Ready

```
VSI MilCAN Rig
  139.184.103.205
    Rules
    Interfaces
    Segments
  139.184.103.201
    Rules
    Interfaces
    Segments
  139.184.103.200
    Rules
    Interfaces
    Segments
      IFACE_CANID0
        node 0x84
        node 0x82
      IFACE_CANID1
      IFACE_CONAPPID
      IFACE_EIPID
      IFACE_EIPID
```

NodeID	ifaceID	rx/tx	MsgID	Priority	Timestamp	LifeTime	StartSlot	PeriodSlots	Description	orgMs...
0x84	0x10	RX	0x5936	4	50676	255	0	100	x/y acceleration	0x5936
0x84	0x10	RX	0x5938	4	50676	255	0	100		0x5938
0x82	0x10	TX	0x5832	4	48883	255	0	10	joystick #2	0x5832
0x82	0x10	TX	0x5830	4	48087	255	0	10	flaps position	0x5830
0x82	0x10	TX	0x3e36	3	48087	255	0	10	gear	0x3e36
0x82	0x10	TX	0x3e34	2	50676	255	0	10	steering wheel position	0x3e34
0x82	0x10	TX	0x3e32	[illegible]	50676	255	0	10	brake pedal	0x3e32
0x82	0x10	TX	0x3e30	2	50676	255	0	10	accel pedal	0x3e30
0x82	0x10	TX	0x4430	3	50676	255	0	10	engine status	0x4430
0x82	0x10	TX	0x5e34	3	50676	255	0	10		0x5e34
0x82	0x10	TX	0x5e32	3	50676	255	0	10		0x5e32
0x82	0x10	TX	0x5e30	5	50676	255	0	10	brakelight	0x5e30

C:\VSISystemdb.mdb

DW_IFACE: 139.184.103.200/Interfaces/10(0.0.0.0)

with 4 Frames per Slot

```
17:05:54 -> Init ok.
17:05:56 -> Populating done. (C:\VSISystemdb.mdb)
17:06:03 -> DW_ROOT: VSI MilCAN Rig Info
17:06:34 -> Bridge 139.184.103.200. Total 26 messages
17:06:34 -> DW_SEGS: 139.184.103.200/Segments Messages
17:06:38 -> Bridge 139.184.103.200, Interface: 0x10. Total 12 messages
17:06:38 -> DW_IFACE: 139.184.103.200/Interfaces/10(0.0.0.0) Messages
```

Figure 4.7: VSI GUI overview

When the configuration is complete the node needs to be taken to operational mode by turning the configuration mode OFF.

4.6 Conclusion

The MilCAN Reconfiguration provides many advantages on maintenance process of a system. It gives the opportunity to a user without any programming knowledge to be able to repair a system and adjust it according to the present requirements. The Node Reconfiguration covers the software side of the maintenance like configuring the message set for every node and upgrading the firmware without the need to utilise the whole network unusable.

The MilCAN Reconfiguration is consisted by three parts, the bootloader that is located in the device's software which then communicates through the Reconfiguration Protocol with the VSI Network Management System that provides control functionalities for all the available operations. Because of the Node Reconfiguration it's possible to connect to different vehicles and access the nodes remotely from a central location.

Chapter 5 MilCAN Fault Tolerance Layer

5.1 Introduction

Safety critical applications depend on networks that provide continuity of service which require redundant network architectures. To provide high level redundancy with COTS CAN equipment is hard. For CAN to become deterministic, MilCAN is introduced which is located between the application layer and the physical layer with no need for any modification to the latest. For the same reason to add continuity of service the Fault Tolerance layer is used, which is located between the application layer and the MilCAN layer.

The design of the MilCAN Fault Tolerant (FT) layer is based on the ability to transparently inter-connect the application layer to multiple buses using a common interface. To achieve inter-operability between all the FT enabled devices proposed MilCAN protocol additions were created. These additions include also the proposed expansion of an already existing MilCAN message for the FT layer use. Additionally, to provide different level of service depending the device and system configuration, the Fault Tolerant operation needs to be broken in to three different blocks. Since MilCAN Fault Tolerance is a software solution, a software protection had to be designed to offer the benefits of a hardware gateway, without the use of any extra hardware.

In this chapter the overall Fault Tolerant layer design is discussed. The individual components for establishing communication between the FT layers of different devices are described, and their functionality is presented. The approach followed to break the FT layer operation into different individual blocks is discussed, along with the additions required to offer Bubbling Idiot protection. [Oikonomidis'08]

5.2 Design

The purpose of the MilCAN (FT) layer is to manage and operate the physical connections of the device to the MilCAN network. The FT layer is transparent to the application layer and manages and is operating the two or more MilCAN buses in order to achieve continuous operation. The application layer is interfaced to the FT layer following a predefined Application Programming Interface (API) which acts as a single virtual MilCAN bus as can be seen in figure 5.1. This allows the use of more than two buses without the affecting the application on the device. The best operational condition is influenced by the working environment of the buses and the physical condition of the network at a given time. In an ideal environment everything should work flawlessly according to the theoretical specification, but in practice and under stressful and demanding conditions this is not the case. Another main design characteristic of the FT layer is that devices that have the FT layer are compatible and can operate with other devices that do not have the FT layer.

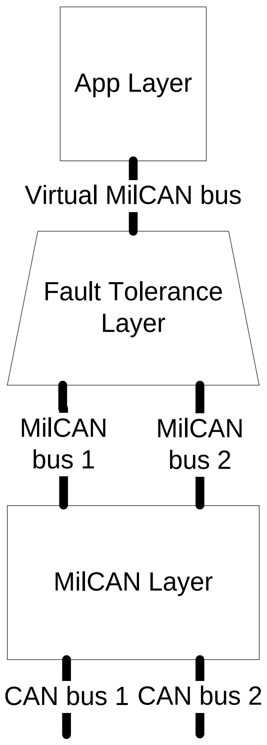


Figure 5.1: MilCAN Fault Tolerant design

Most fault tolerant protocols are using dual bus redundancy to provide more stability on the network. It is therefore proposed as an optional enhancement for MilCAN, depending on the intended application, to have a dual bus where one of the buses can be operating as a primary bus (P-bus) (1Mbps) and the other as a secondary bus (S-bus) (250Kbps, 500Kbps, 1Mbps). In case there is malfunction (physical or data link layers) with the P-bus the S-bus should take over. Because the S-bus is to be used for fail safe solutions it is very important to be able to operate under different conditions. Fail safe is when a component directly reaches a safe state or is brought to a safe state by a special action [Isermann'02]. Where fail silent is when a component exhibits quiet behaviour externally and therefore does not wrongly influence other components [Isermann'02]. A problem that arises through this technique is that when using a bus with four times less bandwidth from the primary, it could result in messages having to be reallocated to new slots in order to keep the same transmitting periods, and the increase in the load would not allow all the messages to be transmitted at the intended time. If this is to be resolved, then smart algorithms should be used in order to filter messages, before they are transmitted by the node, to ensure the reduction of the required bandwidth.

Other fault tolerant systems use bus guardians. The transmitting node informs the bus guardians as soon as the synchronous messages are to be transmitted. This way they are able to know when to allow the node to transmit over the bus. As MilCAN is a purely software based protocol solution on generic CAN hardware, the use of bus guardians will have to be hardware independent. Using the “flooding” methodology at power up and at regular intervals an independent node can monitor each of the buses to detect any possible faults propagating on the bus. In an integrated architecture, this node would be the bridge connecting the dual bus segment. Using this methodology coupled with rescheduling plug-n-play makes fault tolerance possible. [Oikonomidis'09]

5.3 MilCAN FT Layer components

To be able to add the FT functionality to the MilCAN various new components had to be designed. The main new component is the MilCAN FT Master which plays the role of the MilCAN coordinator. It is responsible of the whole MilCAN FT operation and is dynamically selected. The MilCAN FT Master is capable of controlling the rest of the devices with the use of the MilCAN FT Master Frame which is capable of changing various key settings in the FT layer configuration such as bus selection and speed. For the MilCAN FT Master to have an operational overview of the devices that are connected the MilCAN Alive Message has to be extended the required information. The new extended Alive Message is transmitted by all the devices, to inform the FT Master of their status.

5.3.1 MilCAN FT Master

The MilCAN FT Master is responsible on collecting FT information transmitted by the rest of the MilCAN FT devices connected on the same network with the help of the extended MilCAN Alive Message. According to these information the MilCAN FT Master is coordinating the operation of the FT layer on all the devices in the same network. To manage the operational coordination, the FT Master Frame is used. This frame contains the commands that the rest of the devices must follow.

The number of nodes designated as potential MilCAN FT Master within a system is in the responsibility of the system designer. There shall be at least one potential MilCAN FT Master per bus segment. The MilCAN FT Master should not be considered the same as the MilCAN Sync Master. The MilCAN FT Master follows the same election technique as the MilCAN Sync Master but the system designer may choose different potential MilCAN FT Masters from the potential MilCAN Sync Masters.

Following a reset, each device designated as a potential MilCAN FT Master must wait for the receipt of a MilCAN FT Master Frame. If a MilCAN FT Master Frame is not received within a timeout period then this node shall assume the role of MilCAN FT Master and transmit an FT Frame Message. The FT Master Frame timeout period is application specific and is recommended to be double the FT Master Frame period. The system designer should specify the actual value of the FT Master Frame timeout period.

The protocol that will let other nodes to assume the role of the FT Master, must ensure that the potential FT Master with the highest priority will eventually become the FT Master. If the FT Master with the highest priority is non-functional then the next highest priority potential FT Master will become the system FT Master.

5.3.2 MilCAN FT Master Frame

The operation of fault tolerant nodes shall be co-ordinated by the FT Master Frame message. The FT Master Frame is transmitted by the FT Master node only, to all operational buses. Devices connected on the network should adjust their bus configuration according to FT Master Frame. The information that the FT Master frame carries are the master ID, active bus, bus status, bus speed, weight of bus and FT msg counter. The structure of the frame can be seen in table 5.1.

Table 5.1: FT Master frame structure

Header	Payload					
Msg ID	0	1	2	3	4	5
0x0090	Master ID	Active bus	Bus status	Bus speed	Weight of bus	FT msg counter

Master ID Field – this field (payload byte 0) shall be used to identify the Master node that transmits the FT Master Frame. When there is a message been transmitted the source ID address is included in the header of the frame according to the MilCAN specifications. But if there is a bridge or router that retransmits the frame on the bus and changes the header Node ID, the Master node ID in the payload reassures us that the nodes are going to receive the correct message. According to the MilCAN specifications any node can take values from 0x01 till 0xFF, the same restrictions apply for the value of master ID field (table 5.2).

Table 5.2: Master ID field of the FT Master frame

Data	Byte No	No. Bits	Limits/ Range	Comments
Master ID	0	8	0x01 – 0xFF	Can take any value from 0x01 to 0xFF as MilCAN node ID restriction apply.

Active bus field – this field (payload byte 1) shall be used to identify the active bus that carries on the communication. When a device receives a FT Master frame with a different active bus than the one that is currently using, it should switch immediately to the new bus instructed by the FT Master. The value of the active bus field depends on how many MilCAN buses are available on any system configuration. The FT Master decides the best available active bus according to its weighted bus selection algorithm. If the Active bus value is equal to 0 then that means that there none active buses present (table 5.3).

Table 5.3: Active bus field of the FT Master frame

Data	Byte No	No. Bits	Limits/ Range	Comments
Active bus	1	8	0 - 8	Can take any value from 1 to as many available buses are available but less or equal to 8.

Bus status field – this field (payload byte 2) shall be used to identify the status of the available MilCAN buses according to the FT Master. Each bit of the field represents each bus, where the value 1 means operational and value 0 non-operational. For example a system with 3 MilCAN buses where bus 1 is non-operational, bus 2 is non-operational and bus 3 operational the Bus status field will be equal to 4. The purpose of this value is to report the status of the buses at any given time for system monitoring and diagnostic table 5.4.

Table 5.4: Bus status field of the FT Master frame

Data	Byte No	No. Bits	Limits/ Range	Comments
Bus status	2	8	0 - 255	Can take any value from 1 to 255 for 8 healthy buses.

Bus speed field – this field (payload byte 3) shall be used to specify the speed of each bus available in the system. Each bus is represented by 2 bits which can have the following values:

- 1000kbaud = 3
- 500kbaud = 2
- 250kbaud = 1

For example at a system that has 3 MilCAN buses when bus 1 operates at 1000kbaud, bus 2 operates at 500kbaud and bus 3 operates at 250kbaud the bus status field will be equal to 27. When a device receives a FT Master frame and the Bus speed field indicates a different operating speed for the specific bus, then it is required by the device to change to the indicated speed as soon as possible (table 5.5).

Table 5.5: Bus speed field of the FT Master frame

Data	Byte Order	No. Bits	Limits/ Range	Comments
Bus speed	3	8	0 – 255	1000kbaud = 11 (3) 500kbaud = 10 (2) 250kbaud = 01 (1) Can take any value from 0 to 255.

Weight of bus – this field (payload byte 4) shall be used to identify the weight of the transmitting bus. This value is provided by the FT Master after adding the weight value of each active device connected to the specific bus (table 5.6).

Table 5.6: Weight of bus field of the FT Master frame

Data	Byte Order	No. Bits	Limits/ Range	Comments
Weight of bus	4	8	0 – 255	Can take any value from 1 to 255.

FT msg counter field – this field (payload byte 5) shall be a counter that operates in the range 0 to 255. When the counter overflows, it shall be reset to 0. This counter must be used to count how many FT Master Frames have been transmitted in order to detect any messages that have been lost or received in the wrong order. Only the latest message will be kept and older ones will get discarded (table 5.7).

Table 5.7: FT msg counter field of the FT Master frame

Data	Byte No	No. Bits	Limits/ Range	Comments
FT msg counter	5	8	0 – 255	Can take any value from 0 to 255. On reaching the value 255 the counter will be reset to 0 for the next frame.

The minimum transmission frequency of the FT Master Frame by the FT Master shall be 1 Hz. In case of any change in the status of the buses an asynchronous FT Master Frame shall be transmitted in order to continue the normal operation of MilCAN by instantly coordinating the rest of the nodes. The FT Master message must be transmitted on all available operational buses. If the bus is not operational then there should be no attempt to transmit on that bus.

5.3.3 MilCAN Alive Message Extended

All nodes on the bus shall transmit an Alive message to indicate the overall status of the node according to the MilCAN specifications. There have been some modifications on the Alive message payload in order to provide some extra information required for the MilCAN FT operation. The extended version of the MilCAN Alive message is based on the MilCAN Alive

Message with seven extra payloads. Since the structure of the Message is based on the original MilCAN Alive Message, the extended version remains compatible with non FT devices. The Alive message structure share similarities with the FT Master frame except the ones mentioned below (table 5.8).

Table 5.8: Alive Message frame structure

Header	Payload							
Msg ID	0	1	2	3	4	5	6	7
0x62XX	Device status	Master ID	Active bus	Bus status	Bus speed	EWRN flag	Weight of node	msg counter

Msg ID field – this field has primary-type 0x62 and the node ID as a sub-type according to the MilCAN specifications.

Device status – this field (payload byte 0) shall be used to inform the status of the transmitting node on the network. The values that allowed for this payload can be seen in table 5.9. This field is defined by the MilCAN specifications.

Table 5.9: Device status field of the alive message

Data	Byte No	No. Bits	Limits/ Range	Comments

Device status	0	2	0 – 3	Disabled = 00
				Enabled = 01
				Error indicator = 10
				Not available or not installed = 11

Master ID Field – this field (payload byte 1) shall be used to identify the Master node for the specific node which transmits the alive message.

EWRN field – this field (payload byte 5) shall be used to inform about the current status of the Error Warning Status (EWRN) flags of each bus individually. Each bit of the field represents each bus, where the value 1 means EWRN flag on and value 0 EWRN flag off. The EWRN indicates that at least one of the error counters in the EML has reached the error warning limit of 96. EWRN is reset, if both error counters are less than the error warning limit [Infineon'00]. This indication helps in the prevention of any bus failures (table 5.10).

Table 5.10: EWRN flag field of the alive message

Data	Byte No	No. Bits	Limits/ Range	Comments
EWRN flag	5	4	0 – 15	Can take any value from 1 to 15 for 4 healthy buses.

Weight of node – this field (payload byte 6) shall be used to inform the weight of the transmitting node on the network. The FT Master node takes into consideration the weight of every device connected on the network and uses this information in the weighted bus selection algorithm to choose the active bus when there is a bus failure (table 5.11).

Table 5.11: Weight of node field of the alive message

Data	Byte No	No. Bits	Limits/ Range	Comments
Weight of node	6	8	0 – 255	Can take any value from 1 to 255.

The minimum transmission frequency of the Alive message by each node shall be 1 Hz. In case there are changes on the nodes status, the node should transmit asynchronous an Alive message including the new status information. The Alive message must be transmitted on all available operational buses. If the bus is not operational then there should be no attempt to transmit on that bus.

5.4 MilCAN FT Layer operation

The main aspect of the design is the use of multiple hardware platforms and being able to operate on networks with FT enabled devices and FT disabled devices. Because the FT Layer requirements are minimal it is compatible with most CAN enabled devices. Not all the devices on a network are required to have the FT Layer, but it is expected for them to be able to communicate with each other (FT and non-FT). Although they are capable on communicating with each other, in a case of fault the non-FT devices will not be able to guarantee a continuous operation since they are missing the FT capabilities.

The advantages that the MilCAN FT layer provides are separated in to three blocks (figure 5.2): bus error detection, error recovery and bus switching. The error detection and the error recovery are available on all devices using the MilCAN FT layer that meet the MilCAN FT requirements. The bus switching capability is only available on devices that are connected on more than one MilCAN bus that are managed by the MilCAN FT layer. As a result even devices that are connected to a single MilCAN bus have an advantage over the devices that do not have the MilCAN FT layer.

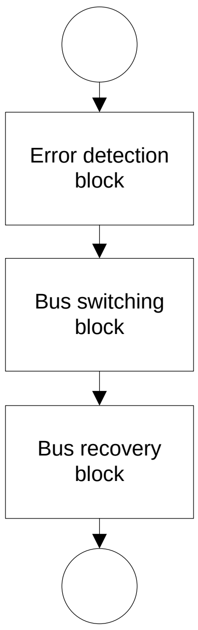


Figure 5.2: MilCAN FT Layer design

5.4.1 Error detection

The Error detection operation is based on two parts. The 1st part is located in every device that has the FT layer and the 2nd part is active only in the potential MilCAN FT Masters. In order for the Error detection of the FT layer to operate properly, these two parts need to be synchronised. For the synchronisation of these parts the Alive Message is used, by informing the potential FT Masters; the current operational status of the devices present in the network. The Alive Message must always be transmitted over all operational buses available. The error detection operation can be seen in figure 5.3.

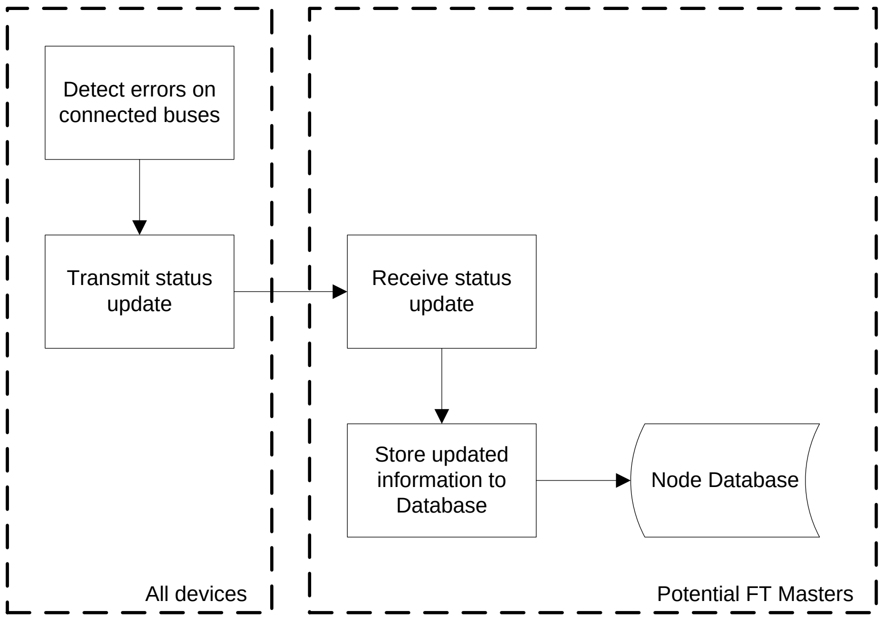


Figure 5.3: Error detection operation

5.4.1.1 Error monitors

To detect errors on the connected buses, the FT layer monitors continuously the CAN controller and the MilCAN layer. The error detection capabilities of the CAN controller are based on the Error Management Logic (EML), Receive Error Counter (REC) and the Transmit Error Counter (TEC) of the CAN controller. When the TEC errors become more than 255 then the CAN controller gets in the Bus Off state, where the controller is disconnected from the bus and become non-operational. When this happens the FT layer immediately detects the change and flags the bus as non-operational and tries to resolve the problem accordingly.

Additionally, there is the bit EWRN in the Status Register, which is set if at least one of the error counters equals or exceeds the error warning limit of 96. EWRN is reset, if both error counters are less than the error warning limit. The EWRN status is included in the Alive message transmitted by the FT enabled devices and helps to forecast potential bus problems. The Weighted Bus Selection (WBS) algorithm at the FT Master node takes the EWRN flag into consideration, when the FT layer is trying to resolve any faults on the bus.

Since the communications is carried with the MilCAN protocol controlled by the MilCAN layer is very important the operation of the MilCAN layer to be overlooked and checked. The MilCAN protocol has three modes: Pre-Operational Mode, Operational Mode and System Configuration Mode. When a device is in *Pre-Operational Mode* means that a valid Sync Frame message is required is order for the device to resume back to normal operation. When a device is in Pre-Operational Mode the MilCAN FT Layer assumes that the bus is not operational and treats the bus as a faulty one. These information are gathered by the FT Layer which is responsible to act accordingly by informing the potential FT Masters with the use of the Alive Message.

5.4.1.2 Node Database

Every potential FT Master device is required to keep a database of all available nodes in the network. In this database the latest status update of each node is stored; to be evaluated by the WBS algorithm to decide the possible available solution. If the MilCAN FT node seize to exist/operate, the next available potential MilCAN FT node will be voted and start acting as the MilCAN FT Master. The new Master needs to be prepared to operate with the least possible delays. To achieve that the Node Database needs to be already populated with the latest status updates. For this reason all potential MilCAN FT Masters need to store the received updates. The data stored on the database, are the data received with the Alive message.

5.4.1.3 Failure Mode and Effect Analysis (FMEA)

To cover any potential failures of a MilCAN network the Failure Modes and Effects Analysis (FMEA) table in table 5.13 was constructed. The FMEA identifies all components, failures, causes, and effects within a system for classification by the severity and likelihood of the failures [Isermann'02]. In table 5.13 are included some of the possible faults that may occur on a MilCAN node and bus. For every fault has specified probability of occurrence and system severity. Also it is specified how these faults are going to be detected, and how to recover them.

Table 5.12: Mishap severity categories

Description	Category
Catastrophic	Class I
Critical	Class II
Marginal	Class III
Negligible	Class IV

Table 5.13: Failure mode analysis results for general purposes

Faults	Failures	Criticality		Detection	Recovery
		Probability of Occurrence	Severity to System		
Cable broken	Non responsive	Medium	Class I	Alive message/ Isolated node status	Use other bus
Cable shorting	Invalid data	Medium	Class I	CAN error counters	Use other bus
Loose cable connection	Invalid data/ Undelivered data	Medium	Class I	CAN error counters/ Alive message	Use other bus
External interference	Invalid data	Medium	Class II	CAN error counters	Bus speed change/ Use other bus
Message corruption	Invalid data	Medium	Class II	CAN error counters	Bus speed change/ Use other bus
Power down	Non responsive	Low	Class I	Alive message	Not defined
Node dysfunction	Babbling idiot	Low	Class I	High frequency transmission	Remote nodes ignore msgs
				Alive msg counter not incremented	Restart/ Shutdown
				Watchdog	
Transceiver dysfunction	Babbling idiot	Low	Class I	High frequency transmission	Restart/ Shutdown
Application software bugs	Application does not refresh data	Medium	Class I	Alive msg counter not incremented	Remote nodes ignore msgs
	Babbling idiot	Low	Class I	Watchdog	Restart/ Shutdown

5.4.2 Bus switching

Depending the system design, if there is more than one bus available, then the Bus switching mechanism must be active. If there is only one bus available then only the Bus recovery mechanism shall be active.

When there is problem with any of the buses, the nodes are responsible to transmit the new updated status to the potential FT Masters. This message should be send through all available MilCAN buses. The Alive message should contain the up-to-date status of the node and the value “0” for the active bus field. Upon reception of the Alive message only the FT Master should coordinate the switch, with the use of FT Master Frame. The FT Master must decide the next active bus according to the results from the WBS algorithm.

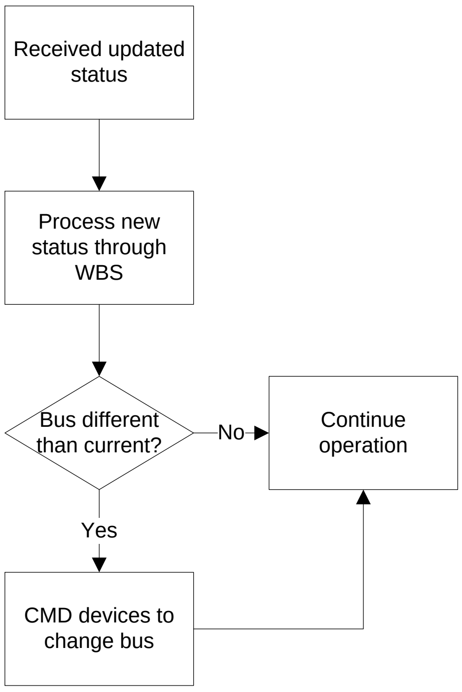

Received updated status
Process new status through WBS
Bus different than current?
No
Continue operation
Yes
CMD devices to change bus

Figure 5.4: Bus switching - MilCAN FT Master device

When the result of the WBS algorithm is different than the current active bus, then the FT Master node commands the FT Slave nodes to change to that bus by sending the MilCAN FT Master Frame. The transmission of this frame will be asynchronous over all the available buses and will be directly after WBS came with the new result. If there is no change on the active bus then there must be no asynchronous transmission of the FT Master Frame. The FT Master Frame is periodic during normal operation, to prevent any synchronisation issues that may arise between the devices during the switching operation. It also prevents any devices that powered on or reset to start using the correct bus. The operation of the FT Master device can be seen in figure 5.4.

When a FT Slave device receives the command (FT Master Frame), it switches to the new bus if different from the current one. The switch must be immediate if the suggested bus is operational, otherwise it should remain to the previous bus if operational. By doing so, in case the bus is split, the device will be able to reach a smaller number of devices instead of none. The operation of FT Slave devices can be seen in figure 5.5.

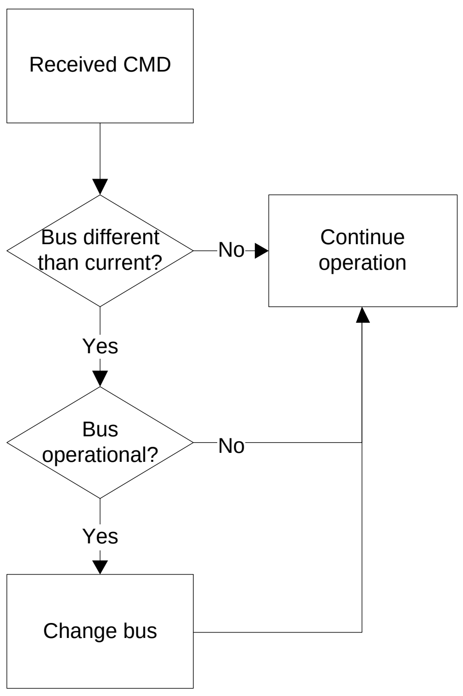


Figure 5.5: Bus switching - MilCAN FT Slave device

5.4.2.1 Weighted Bus Selection (WBS) algorithm

When according to the predefined conditions there is a need to change the active bus then the weighted bus selection algorithm determines which bus will be the next active. This selection follows the conditions at this specific order (figure 5.6):

1. Importance of nodes connected on the bus (weight of node).
2. Level of errors on the bus (EWRN).
3. Operating speed of the bus.

At any given time all the potential FT Master nodes are responsible to be aware of the above information.

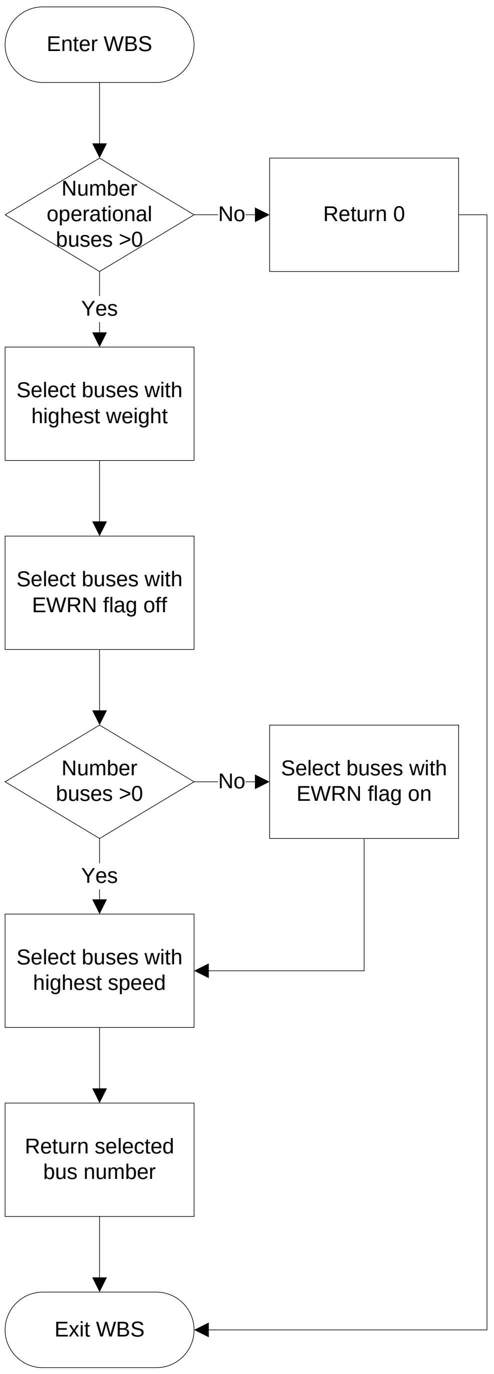

Enter WBS
Number operational buses >0
No
Return 0
Yes
Select buses with highest weight
Select buses with EWRN flag off
Number buses >0
No
Select buses with EWRN flag on
Yes
Select buses with highest speed
Return selected bus number
Exit WBS

Figure 5.6: Weighted Bus Selection

1. The importance of the nodes is defined by the system designer by assigning different weight values to every node, with the highest value to represent the highest priority. In case of emergency and with no fully operational buses available to the majority of the nodes, the bus with the highest score will be selected. The system will continue its operation with that till another bus is detected / recovered, where the weighted bus selection algorithm will determine if the active bus should change.

2. The level of errors of the future active bus must be low in order to assure that there are less chances of the bus under heavy load to become faulty. The EWRN flag indicate if there will be a rise of errors on the bus, which will help the WBS to choose a more suitable bus.
3. The system designer may choose the secondary buses to operate on different bus speeds or to have variable bus speeds which will change during the recovery process. Higher priority is given to the buses with higher operational speed and lower error level.

5.4.3 Error recovery

When a bus becomes non-operational, the FT layer tries to recover it back to fully operation by initiating the bus recovery mechanism. During this process the following procedure should be followed.

1. Disable MilCAN on that bus.
2. Enable MilCAN on that bus.
3. Inject test frames on the bus and expect response from the FT Master. (ping-pong)
4. Monitor error level on the bus.

To follow the above procedure and archive an operational bus that will be capable on working properly at any conditions, the error recovery mechanism is broken down to three mechanisms. In case more than one bus needs to be recovered then the Bus recovery mechanism will operate on all the faulty buses at the same time. The operation of the error recovery can be found in figure 5.7.

5.4.3.1 Bus restart

When the number of errors in the Transmit error counter becomes higher than 255 then the CAN controller goes to a Bus off state where is completely disconnected from the bus. To get the CAN controller back to the Error active state, it needs to be restarted. The MilCAN FT layer is controlling this operation to revert the bus back to operational mode. To restart the CAN controller, it restarts MilCAN for the specific bus by deactivating it and activating back again. During this procedure the MilCAN bus is reinitialised. Between the deactivation and activation the operational speed of MilCAN may change according to the bus speed change part of the Error recovery algorithm. It is suggested that the time delay between the deactivation and activation should be 200ms, but it is up to the system designer to decide any other more appropriate value for a specific system.

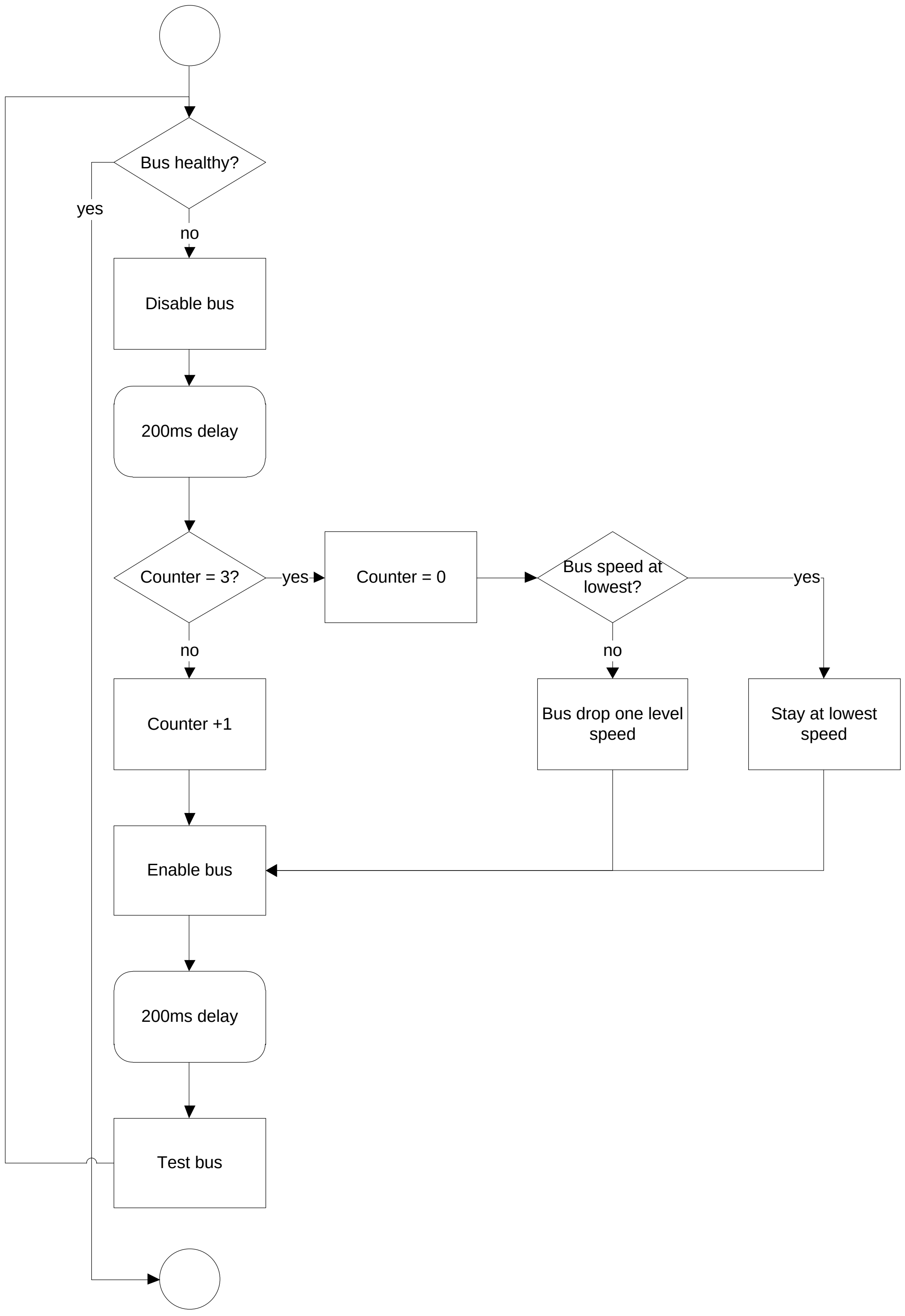


Figure 5.7: Error recovery block

5.4.3.2 Bus speed change

When the bus is not recoverable then the bus recovery mechanism could try to drop the operational speed of the bus. When the bus is operates under lower speed it becomes less sensitive on external interferences as a result the bus can become operational again by working at lower bus speeds. Every three restart attempts the bus should drop speed. This is up to the system designer, depending the operation of the system.

It is very important to have a synchronised operation between all the nodes connected to the bus. All the nodes connected to the specific bus need to change to the same operating speed. When a node is operating on different bus speed then it will introduce error frames on the bus. To avoid this, the FT Master is responsible to transmit the speed change command to all the connected nodes. The transmissions should happen through all available healthy buses with the use of the FT Master Frame. To avoid any missed transmissions, the message needs to be transmitted three consecutive times. This covers the case that the nodes that are connected to the faulty bus have at least one other fully operational bus that will receive the command messages from.

In the case that the node is not connected to any other healthy bus or is operating on a single bus, the bus speed must not be changed. If the devices start rotating to the available operating speeds, they will get unsynchronised and impossible to find a common operational speed between all of them. It is suggested for some configurations the system designer must configure one bus always to operate at a low constant speed to be used as a backup bus and for the FT Master Frames to be transmitted from.

5.4.3.3 Scheduling

In order to recover the bus back to operational mode the CAN speed may change, so the bus becomes less sensitive. However, this could affect the scheduling of the application, because at lower speeds the MilCAN cycle becomes longer and the messages that are synchronous and assigned to specific slots will be transmitted with delays since MilCAN messages are scheduled according the sync frame. If there is going to be a bus speed change then the schedule should change also to ensure real time communication. To overcome this problem there are various options and is up to the developer to choose which one fits him better. The system designer has three options to follow in case of bus speed drop:

- Use different schedules for every given CAN speed hardcoded.
- Remain with the same schedule as in normal operation speeds.
- Increase the frequency of HRT MilCAN frames and drop the frequency of the SRT MilCAN Frames.
- Increase the frequency of HRT MilCAN frames and drop completely the transmission of the SRT MilCAN frames.

This way there will be bandwidth available for the higher priority messages. If the operation of a node is not so important, it can stop transmitting messages completely. Any kind of solution adopted has to make sure that the limit of the bandwidth of the bus is not exceeded. There is a solution for every operational condition.

5.5 Babbling idiot

In a distributed hard real-time system based on a broadcast bus for inter-node communication it is important to prevent a single faulty node from monopolizing the communication bus. In a time-triggered system, in which messages are broadcasted according to a pre-determined transmission pattern, this kind of failure is characterized by the faulty node transmitting messages at arbitrary points in time thus corrupting the transmissions on the bus. This type of failure is known as the babbling idiot failure [Temple'98].

The use of a bus guardian added to each node to protect the communication bus from the babbling idiot failure, is usually used by safety critical networks. Since the MilCAN FT Layer is a software solution, the proposed solution has to be software based. The regular transmission pattern of a time-triggered system is exploited in order to enforce a fail-silent behaviour of the node in the time domain. Using fail-silent nodes greatly reduces the complexity of designing distributed fault-tolerant systems.

A node is considered to be fail-silent if it exhibits the following behaviour [Temple'98]:

- The node sends correct messages at specified points in time, that can be verified as being correct by all non-faulty receivers.
- The node sends corrupt messages at specified points in time, that can be identified as being corrupt by all non-faulty receivers. These messages are discarded.
- The node sends no messages at all.

5.5.1 Message filtering

By using a message filtering system, devices that start acting as babbling idiots are tried to be recovered and if that fails then it will stop any communication through MilCAN. To manage that, each node has a list of the transmitted messages on the system and their transmission frequency. When the frequency of received messages is higher than the expected one, the transmitting node is treated as a Babbling Idiot (BI). When the received message is not included in the list, its frequency has to be less than the default one assigned by the system designer. When a babbling idiot is detected by a node, it is reported on all healthy buses by that node using the **FT BI report** message. The FT master then collects the reports from the nodes and analyses them. According to this information it determines if the problem is on the originator of the messages or in the node that reported the error. The node is then warned by the FT Master by transmitting the **FT BI warning** message and will try to recover by itself by running a recovery routine. The recovery routine is custom for every node and assigned by the system designer. When the node is still not operating properly after being warned for a specific number of times, the FT Master transmits the **FT BI hard-reset** message. If even after the hard-reset command the node is not operating properly, then the FT Master transmits the **FT BI hard-kill** message. When a node receives the hard-kill command, it should manually shutoff MilCAN, interrupts and the application. Also as an extra safety measure any messages received from a babbling idiot node, must not been forwarded to the application layer.

5.5.2 FT BI report

The FT BI report is transmitted by any node, to all operational buses. The FT Master node collects these reports and after analysing them, determines which node is the Babbling Idiot.

Table 5.14: FT BI report

Header	Payload	
Msg ID	0	1
0x0094	Node ID	BI node ID

BI node ID Field – this field (payload byte 1) shall be used to identify the babbling idiot node.

Table 5.15: Master ID field of the FT Master frame

Data	Byte No	No. Bits	Limits/ Range	Comments
BI node ID	1	8	0x01 – 0xFF	Can take any value from 0x01 to 0xFF as MilCAN node ID restriction apply.

5.5.3 FT BI warning

The FT BI warning is transmitted by the FT Master node only, to all operational buses. When a node with matching BI node ID receives this message, executes the recovery routine that has been assigned by the system designer.

Table 5.16: FT BI warning

Header	Payload	
Msg ID	0	1
0x0091	Master ID	BI node ID

5.5.4 FT BI hard-reset

The FT BI hard-reset is transmitted by the FT Master node only, to all operational buses. When a node with matching BI node ID receives this message, should reset itself.

Table 5.17: FT BI hard-reset

Header	Payload	
Msg ID	0	1
0x0092	Master ID	BI node ID

5.5.5 FT BI hard-kill

The FT BI hard-kill is transmitted by the FT Master node only, to all operational buses. When a node with matching BI node ID receives this message, should manually shutoff MilCAN, interrupts and the application.

Table 5.18: FT BI hard-kill

Header	Payload	
Msg ID	0	1
0x0093	Master ID	BI node ID

5.5.6 Watchdog

The microcontroller's watchdog is responsible to reassure a continuous error free operation of the controller. The watchdog is used to detect any application malfunctions during the operation of the node. The use of the watchdog is suggested but it is up to the system designer to decide. The FT layer is responsible to reset the watchdog during its operation on predefined intervals. If it fails to do so, the watchdog will reset the hardware of the microcontroller. When a hard-kill command is received by the node, the watchdog has to be disabled. By using the watchdog, the possibility that the device does not receive any MilCAN messages in order to be controlled by the FT Master is eliminated.

5.6 Conclusion

Within this chapter the overall design of the MilCAN Fault Tolerance Layer is discussed from the theoretical and conceptual perspective. There is a detailed explanation of the FT layer design, and how it operates. It also includes the MilCAN FT Layer vital components which are the communication messages used between MilCAN FT devices. The operation of the layer is analysed as whole and broken down to block responsible for different parts of the FT operation. Last is discussed how to solve the Babbling Idiot problem without the aid of hardware support.

The development of a FT layer located between the Application layer and the MilCAN layer is a key contribution to the development of the High Availability MilCAN. This standardisation expands the use of Multiple MilCAN buses seamlessly without affecting the application layer. Furthermore, an expansion is introduced to the MilCAN protocol allowing the communication between the FT layers of all connected devices. The MilCAN FT layer operation mechanisms are presented separately, task dependant. These mechanisms are compatible with devices managing one or more MilCAN buses and can coexist with non FT devices. Additionally a software solution has been presented, how to resolve babbling idiot problems without the use of extra hardware. This solution has introduced an additional set of instructions that are used for the communication of the devices to resolve babbling idiot issues.

Chapter 6 Testbed for vetronics evaluation and verification

6.1 Introduction

This chapter the development testbed and VSI testbed are introduced. The High Availability MilCAN is tested and evaluated on two different testbeds. The components compromising the complete systems are listed and investigated, to clearly identify the capabilities of the platforms along with its current status and operations.

The High Availability MilCAN has been developed and evaluated using the testbeds, starting from its initial construction phase following to the current setup. As a core part of the testbed, the High Availability MilCAN is benchmarked with the operational conditions that are within the limits of the testbeds to be simulated. The individual systems that are part of the testbed add to the overall “complexity” of the model created, upon which the testing vectors are applied.

The measurements gathered for the performance are based on various traffic scenarios that are programmed to the system to show the response of the High Availability MilCAN operation in term of message latencies. The operation of the MilCAN Fault Tolerance has been evaluated based on various fault scenarios that were introduced in to the system to show the reaction of the MilCAN High Availability.

6.2 Development testbed layout and components

The development testbed is a single MilCAN segment connected to two desktop computers. It provides a controlled environment appropriate to be used during the development and later to verify the operation of the system. The testbed is constructed by using of the self equipment.

6.2.1 Hardware devices

The testbed includes devices that can be categorised as embedded systems and computer systems. The embedded systems are four microcontrollers and the computer systems are two workstation PCs.

Table 6.1: Infineon C167CS specifications

Family	Siemens C166
Arch	16bit RISC
Core speed	20MHz
FLASH	128kb
RAM	64kb
Features	Dual CAN (onchip)

The embedded development boards are phyCORE167 platforms bearing the Infineon C167CS processor (C167CS, table 6.1). The computer platforms consist primarily of commercial workstations, their specifications range from low-profile embedded systems to high- end systems (table 6.2).

Table 6.2: Computer systems specifications

Operation	CANoe	Data login
Family	i386	
Arch	32bit	64bit
CPU	Pentium 4	Quad Core 2
FPU	Yes	
Core(s) speed	1.70GHz	2.40GHz
Memory	1GB	4GB
Memory speed	133MHz	333MHz
Flash	N/A	
Network	10/100Mbit-FDX Ethernet	

The *CANoe* computer is connected straight on the two CAN buses with a CANcardXL (CAN card). Its purpose is to collect data from the two CAN buses and display them. The *Data login* computer is connected straight on the microcontrollers with the serial RS-232 interface. The microcontrollers export vital operational information through the serial port and the *Data login* computer collects and display them. The configuration and layout can been seen on figure 6.1. The *Data login* computer is also connected on the two CAN buses with a CANstressDR in order to introduce problem and errors on the buses.

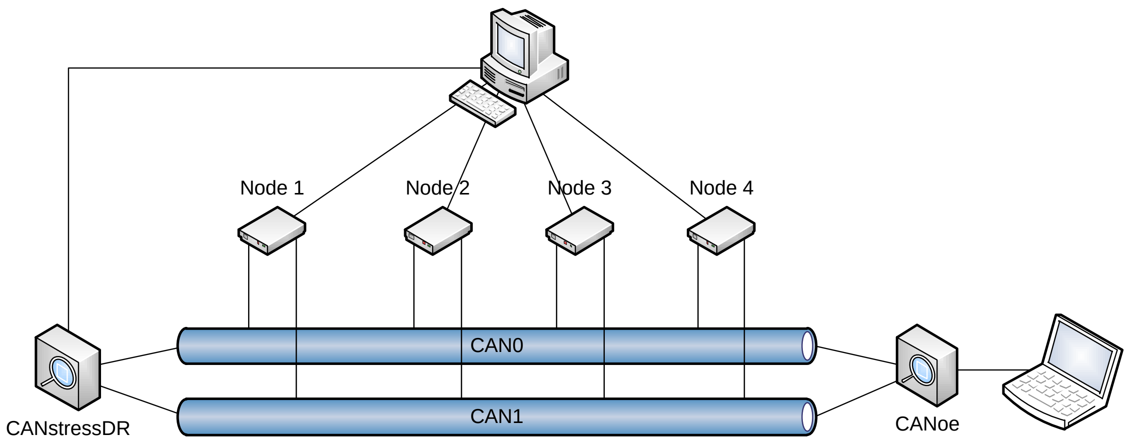


Figure 6.1: Development testbed topology

6.3 VSI testbed components and layout

The Vetronics Research Centre (VRC) prepared a testbed that is a platform that evaluates vetronic and embedded networks. It is being used to develop, test and demonstrate current and future technologies. The testbed accommodates three different segments Automotive, Utilities and Multimedia which are based on MilCAN. The three segments are interconnected through MilCAN and Ethernet using the VSI Bridge which is configured through the VSI GUI. For the VSI testbed custom software and hardware has been developed, except the large number of off-the-shelf hardware that has been used.

6.3.1 Hardware devices and components

The testbed includes devices that can be categorised as embedded systems and computer systems. For demonstration purposes a remote controlled vehicle which is connected wirelessly to the testbed is developed and built. A large number of the hardware components are off-the-self, but also specialised hardware has been developed by the VRC for the VSI Bridge.

6.3.1.1 Embedded systems

The embedded development boards are the phyCORE167 platforms bearing the Infineon C167CS processor (C167CS, table 6.1). The devices are modified to support in-cable power provided by the MilCAN bus. The modification provides DC-DC voltage regulation and filtering. The two nodes attached to the VSI Bridges are based on the C167CS-USB.

6.3.1.2 Computer systems

The computer systems used for the testbed, consist primary of of-the-self workstations and the specification range from low to high end systems as can be seen in table 6.3. The BRG-E100LX and BRG-ITX are connected to the C167CS-USB and are responsible to interconnect the MilCAN segments and the backbone. These computers are running instances of the VSI Bridge which is remotely configured by the VSI GUI. The Crew station is an industrial computer equipped with a touch-screen display. Also two additional displays are attached to the computer. It is used to display the VSI GUI and video stream coming from the Video server. The Video server is a computer responsible in capturing, encoding and serving the two stream input videos. The two sources are a USB webcam with pan and tilt capabilities and a S-Video input. The video are encoded and streamed to the clients.

Table 6.3: Computer systems specifications

Operation	BRG-E100LX	BRG-ITX	Crew station	Video server
Family	CRIS	i386		
Arch	32bit			
CPU	100LX	C3 Nemiah	Athlon XP	Pentium 4
FPU	No	Yes		
Core(s) speed	100MHz	1GHz	1.47GHz	1.8GHz
Memory	32MB	128MB	256MB	256MB
Memory speed	50MHz	133MHz	133MHz	133MHz
Flash	8MB	N/A		
Network	10/100Mbit-FDX Ethernet			

6.3.1.3 Remote controlled vehicle

For the visual demonstration of the testbed a remote controlled vehicle controlled via motor controllers through an on-board C167CS was constructed. The C167CS is connected through a RF link to a node in the testbed which is responsible to transmit the control commands. The control commands generated from the testbed are transmitted to the remote controlled vehicle. An on board camera with pan and tilt functionalities is transmitting the video to the video server which is transmitted to the crew station. Since the remote controlled vehicle is battery powered, it can be completely independent and remotely controlled. It can move forward, backwards and turn left and right. All these controls are transmitted from the steering wheel and pedals devices on the testbed. Additionally the camera movement is controlled by the joystick controller attached on the steering wheel. In case of signal loss the vehicle halts and waits for the next valid command.

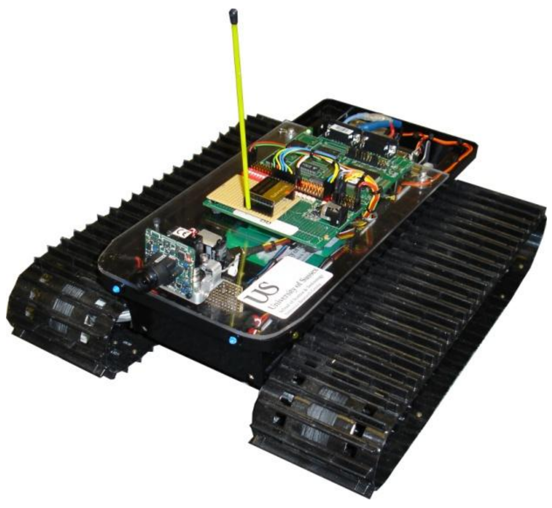


Figure 6.2: Remote controlled vehicle

6.3.1.4 Cabling

For the physical MilCAN network connections a coaxial cable with five wires is used. One pair is used by CAN data lines, one pair for power, and a separate cable for earth grounding. Following the MilCAN specifications the two pairs, data and power are individually shielded and with an overall shield for the whole cable. The connectors used are manufactured for DeviceNet applications, which is a CAN based protocol. To connect the segments to the backbone T-piece connectors are used, and for the nodes in the segments drop-boxes which are connected only by power to the backbone. The nodes are connected to the drop boxes through custom cables that convert the DeviceNet screw terminal to a DB9 plug.

6.3.2 Network layout

The VSI testbed has three MilCAN segments grouping the MilCAN devices according their functionality as can be seen in figure 6.3. The three segments are:

- Utilities segment (SEG-U)
- Automotive segment (SEG-A)
- Multimedia segment (SEG-M)

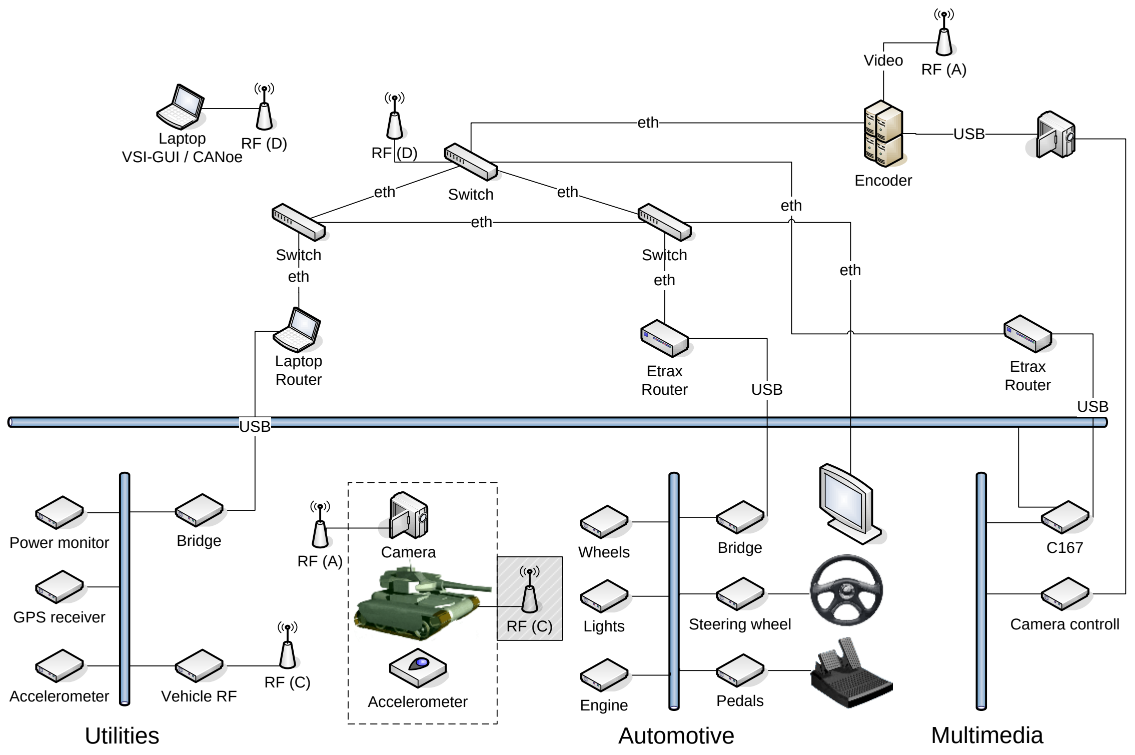


Figure 6.3: VSI testbed layout

All segments are connected to the MilCAN backbone to provide power to the segments with a power only connection bit the CAN bus lines are not connected. SEG-U and SEG-A are connected to the MilCAN and Ethernet backbone with the use of VSI Bridges. SEG-M is only connected to the MilCAN backbone through a node acting as a MilCAN-to-MilCAN cut-through bridge.

6.3.2.1 Utilities segment

Electronic devices that are not related to the motion or control of the vehicle are placed in the Utilities segment. The nodes included in the Utilities segment are:

- Power monitor
- GPS
- Accelerometer
- Vehicle RF

The **Power monitor** node is responsible to measure the amount of power being drawn by the MilCAN devices by using an external current transducer. It is important for vehicle systems to be able to measure the current consumption since electronics may drain the battery of the vehicle.

The **GPS** node gets the exact coordinates from the GPS receiver which is connected through serial interface to the node. The protocol that has been used for the communication with the device is NMEA.

The **Accelerometer** monitors vibration and acceleration off the vehicle, a scheme that allows the prediction of possible mechanical failures due to the extreme operating conditions. The accelerometer and the power monitor node are responsible for the internal monitoring of the vehicle.

The **Vehicle RF** node acts as a link with the remote controlled vehicle which is used to demonstrate the operation of the vehicle network. The node receives the control messages from the steering wheel, and pedals nodes and retransmits them to the remote controlled vehicle. The vehicle transmits to the testbed accelerometer readings taken by the remote controlled vehicle sensors. The communication is established by the use of a custom bidirectional communication protocol through digital RF transceivers. This protocol is bidirectional allowing the control of the vehicle but also the collection of data from the on-board sensors on the vehicle.

6.3.2.2 Automotive segment

Devices that are directly related with the movement of the vehicle are placed in the Automotive segment. The nodes included in the Automotive segment are:

- Wheels
- Lights
- Engine
- Steering wheel
- Pedals

The **Wheels** and **Lights** are demonstrated by virtual means (LEDs) which show the status of the nodes. The Wheels node represent the wheel movement of the vehicle is displayed with the use of a row of LEDs that indicate the current wheel position. The Lights node shows the state of the head lights, break lights and indicators by turning the representing LEDs ON/OFF.

The **Engine** node represents the throttle control on the engine. For demonstration purposes the node is connected to an external speaker where the output sound changes according to the throttle position and gear selected. Low revolutions are represented with a low frequency output, where high revolutions are represented with a high frequency.

The **Steering wheel** and the **Pedals** are essential for controlling the vehicle. The nodes are connected to external pedals and a steering wheel. The steering wheel also includes a set of buttons, joysticks and flaps. The buttons are used to control the lights on the vehicle; the joysticks control the camera movement on the remote vehicle and the webcam located on the testbed. The flaps control the gears as can been seen in semi-automatic vehicles.

6.3.2.3 Multimedia segment

Devices that are oriented towards the control of audio and visual applications are placed in the multimedia segment. The nodes included in the Multimedia segment are:

- Camera control
- Dot Matrix Display

The **Camera control** node is connected to the web camera and can give the commands to move the camera up/down and left/right. The commands are received from the steering wheel and are relayed to the webcam through a serial link.

The **Dot Matrix Display** node is connected to a Dot Matrix monochrome display unit. This displays status information from the sub-systems of the testbed such as GPS coordinates, pedals status, engine status, steering wheel direction and power consumption of the system.

6.3.2.4 Backbones

The three segments in the testbed intercommunicate through two backbone networks which are a MilCAN bus and a high-speed (100Mbit) full-duplex switched Ethernet. The MilCAN network offers deterministic operation and the Ethernet offers high-speed networking. The MilCAN backbone operates to speeds up to 1Mbit and it is terminated in both ends without reaching the maximum length of 40 meters. The operational speed for the Ethernet backbone is 100Mbit. The centre of the Ethernet backbone is a full-duplex Ethernet switch where a number of devices are connected to it such as VSI Bridges, crew station, video server and a WiFi access point. [Charchalakis'03]

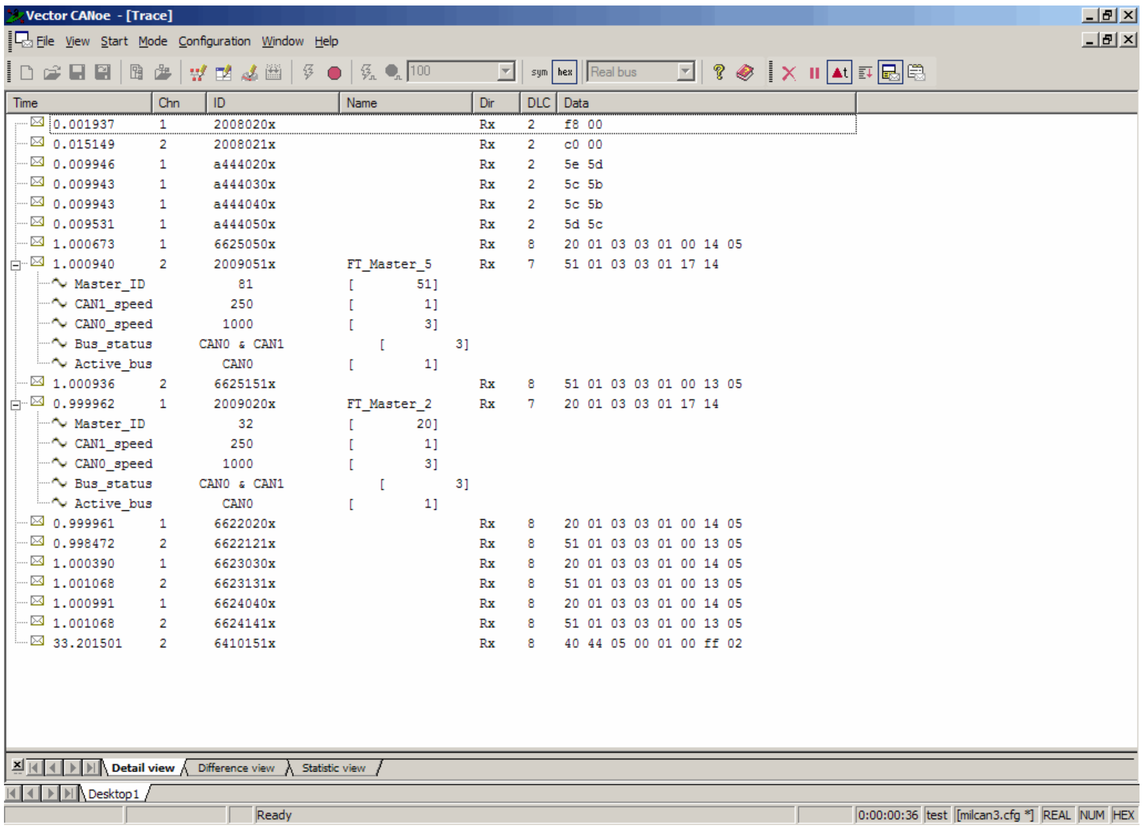
6.4 Software

Both the development and VSI testbeds required some extra custom or commercial software to be used during the evaluation and measurement period. To analyse the testbed operation CANoe from Vector is used and to introduce faults and disturbances CANstress also from Vector is used. To analyse the log file output taken from CANoe and determine message latencies a custom software written in C++ is developed. In order to evaluate the testbed operation a real time monitoring system is developed. This system consists of three parts, the status transmission system located in the devices with the *Fault Tolerant layer*, the raw status collector located in the *Data login* computer and the status GUI used to display the status of the connected devices with the help of a GUI.

6.4.1 CANoe

CANoe is an all-round tool for the development, testing and analysis of entire networks. It supports the user during the entire development process; from planning to start-up of entire distributed systems. CANoe’s versatile functions and configuration options are used by network designers, development engineers and test engineers at OEMs and suppliers. The CANoe user can test and analyse the multi-bus communication and complete systems at the development work place, during the system integration as well as in the vehicle. By using hardware data logger during test drives, the logged bus traffic can be evaluated with all CANoe functions at a later date [Vector'06a].

CANoe was used all the way through the development and measurement stages and it was of great aid. As can be seen in figure 6.4 CANoe is capturing the raw CAN data transmitted on the two buses. While capturing the columns that are of interest are the Time, Chn, ID, Name, DLC and Data.



```
Time          Chn   ID           Name          Dir   DLC   Data
0.001937      1     2008020x                   Rx    2     f8 00
0.015149      2     2008021x                   Rx    2     c0 00
0.009946      1     a444020x                   Rx    2     5e 5d
0.009943      1     a444030x                   Rx    2     5c 5b
0.009943      1     a444040x                   Rx    2     5c 5b
0.009531      1     a444050x                   Rx    2     5d 5c
1.000673      1     6625050x                   Rx    8     20 01 03 03 01 00 14 05
1.000940      2     2009051x     FT_Master_5   Rx    7     51 01 03 03 01 17 14
  Master_ID           81         [        51]
  CAN1_speed          250        [         1]
  CAN0_speed          1000       [         3]
  Bus_status       CAN0 & CAN1       [          3]
  Active_bus          CAN0       [         1]
1.000936      2     6625151x                   Rx    8     51 01 03 03 01 00 13 05
0.999962      1     2009020x     FT_Master_2   Rx    7     20 01 03 03 01 17 14
  Master_ID           32         [        20]
  CAN1_speed          250        [         1]
  CAN0_speed          1000       [         3]
  Bus_status       CAN0 & CAN1       [          3]
  Active_bus          CAN0       [         1]
0.999961      1     6622020x                   Rx    8     20 01 03 03 01 00 14 05
0.998472      2     6622121x                   Rx    8     51 01 03 03 01 00 13 05
1.000390      1     6623030x                   Rx    8     20 01 03 03 01 00 14 05
1.001068      2     6623131x                   Rx    8     51 01 03 03 01 00 13 05
1.000991      1     6624040x                   Rx    8     20 01 03 03 01 00 14 05
1.001068      2     6624141x                   Rx    8     51 01 03 03 01 00 13 05
33.201501     2     6410151x                   Rx    8     40 44 05 00 01 00 ff 02
```

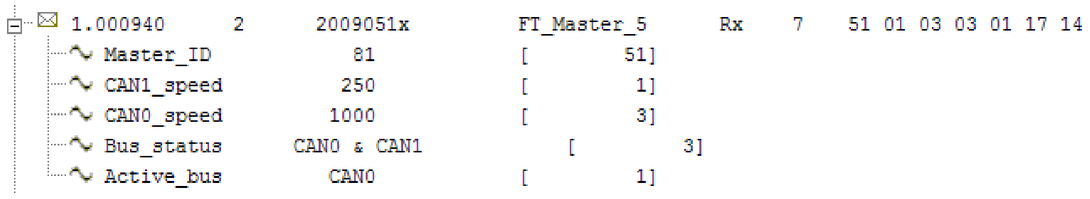
Figure 6.4: CANoe User Interface

Time - What will be displayed in the Time column is configurable between two choices the time passed since the beginning of the capturing and the time passed since the last message received with the same ID.

Chn - Represents the channel number of the CANoe probe. As can be seen in figure 6.4 there are messages received on channel 1 and 2.

ID - Since CANoe is not support MilCAN specifically, the displayed information are treated as CAN data. In the ID field can be seen the whole header of the MilCAN frame where the priority, primary ID, secondary ID and other MilCAN header information are included.

Name – In CANoe you are allowed to assign names on predefined messages and process the real-time data received according to your configuration. Figure 6.5 shows the **FT_Master_5** message where individual message characteristics are analysed and presented in an easier understandable format. As is shown in the figure the **CAN0_speed** is 1000 which is being derived from the data of the message.



```
1.000940      2      2009051x        FT_Master_5      Rx    7    51 01 03 03 01 17 14
  Master_ID           81           [          51]
  CAN1_speed          250          [           1]
  CAN0_speed          1000         [           3]
  Bus_status        CAN0 & CAN1          [           3]
  Active_bus          CAN0         [           1]
```

Figure 6.5: CANoe message defining

DLC – The DLC column shows how many bytes are transmitted in the payload of the specific message.

Data – In the data column the actual data that are being received and are shown separated in byte sizes.

Except capturing data from the CAN buses CANoe has the ability to inject data to the bus. The triggering for the injection can vary; the one used during testing was with the press of a button. Different messages were assigned to different buttons on the computer keyboard. These messages where used to operate the testing firmware when collecting performance measurements.

6.4.2 CANoe Log analyser

While capturing data with CANoe, the collected data except being displayed real-time, they can be also saved in a file in a space separated format. When measuring the performance of the High Availability MilCAN, the latencies of messages being transmitted and received were measured. To be able to have as accurate results as possible, thousands of messages were captured by the two probes of CANoe. In order to be able to find the time difference between thousands of transmitted and received messages, a custom software that was able to read the CANoe file format, identify the messages and calculate the time difference was required.

This software is programmed with the C++ programming language and does not have a GUI. The input will accept a CANoe log file and the output will be a file with the calculated results. While creating the software, it was realised that C++ is not the most appropriate language for mathematical calculations with numbers that have many decimal places. For that reason, decimal numbers have been multiplied with 10.000 to convert it to an integer number. This way the calculations that follow provide an accurate output that is not affected by the C++ decimal issues.

6.4.3 CANstress

CAN networks are highly tolerant in respect of disturbances of the bus communication and failures. In order to test whether a system is behaving properly in case of disturbances or failures, a device used to disturb the CAN bus, its physical properties and the logical levels (recessive and/or dominant) in a targeted, reproducible way. CANstress is a standalone hardware module that is inserted directly onto the CAN bus. It contains various triggering conditions and disturbance logics.[Vector'06b]

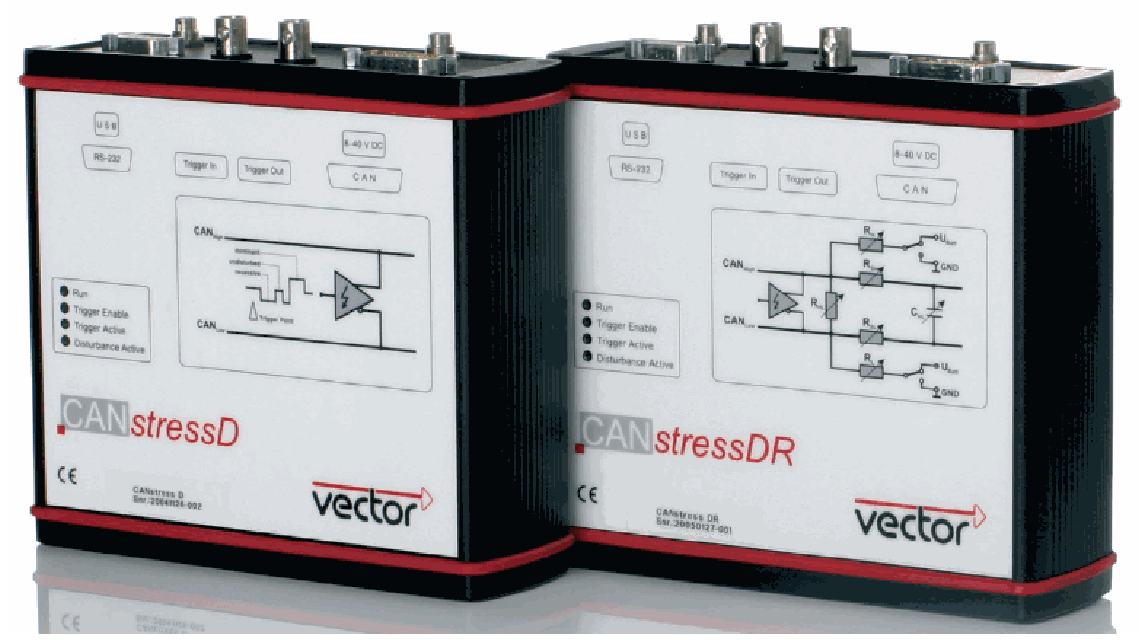
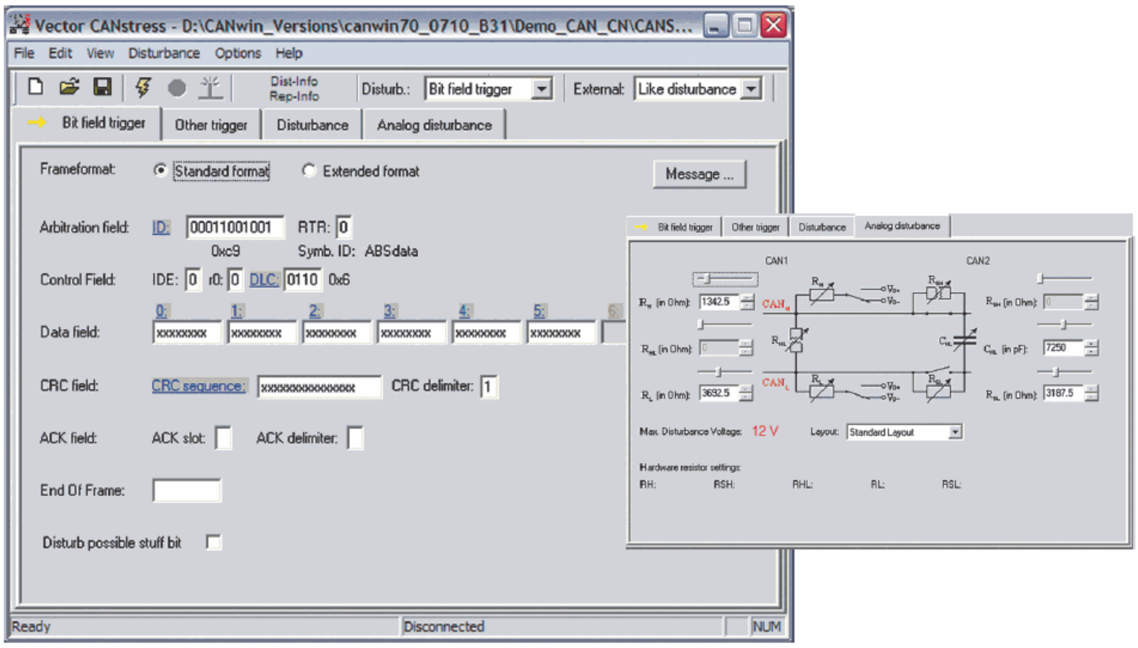


Figure 6.6: CANstress device


Vector CANstress - D:\CANwin_Versions\canwin70_0710_B31\Demo_CAN_CN\CANS...

File Edit View Disturbance Options Help

Dist-Info Rep-Info | Disturb.: Bit field trigger | External: Like disturbance

Bit field trigger | Other trigger | Disturbance | Analog disturbance

Frameformat: [✓ Standard format] Extended format | Message ...

Arbitration field: ID: 00011001001 RTR: 0
0xc9 Symb. ID: ABSdata

Control Field: IDE: 0 r0: 0 DLC: 0110 0x6

Data field: 0: xxxxxxxx 1: xxxxxxxx 2: xxxxxxxx 3: xxxxxxxx 4: xxxxxxxx 5: xxxxxxxx 6:

CRC field: CRC sequence: xxxxxxxxxxxxxxx CRC delimiter: 1

ACK field: ACK slot: ☐ ACK delimiter: ☐

End Of Frame:

Disturb possible stuff bit ☐

Ready | Disconnected | NUM

Bit field trigger | Other trigger | Disturbance | Analog disturbance

CAN1 | CAN2

R_H (in Ohm): 1342.5
R_{HL} (in Ohm): 0
R_L (in Ohm): 3692.5
R_{SH} (in Ohm): 0
C_{HL} (in pF): 7250
R_{SL} (in Ohm): 3187.5

Max. Disturbance Voltage: 12 V Layout: Standard Layout

Hardware resistor settings:
RH: RSH: RHL: RL: RSL:

Figure 6.7: CANstress configuration software

In figure 6.6 the actual CANstress device is shown, it has a USB connector that is then connected to the computer. On the computer using the CANstress configuration software (figure 6.7) the user is capable to configure the CANstress device to introduce specific disturbances and when to be triggered. There is a wide range of faults that can be generated with the device, which proves very helpful in order to test the MilCAN Fault Tolerant Layer for various scenarios and cases.

6.4.4 Real-time monitoring system

During the development period of the High Availability MilCAN it was vital to have a real-time monitoring / debugging system. The compilers that were used to develop the software for the devices provide such functionality, but with serious performance and compatibility issues. Because of that, a custom high efficiency real-time monitoring system is developed. The goal of this system was to transmit through the serial port data in real time. Also it has the option to store these data temporary in memory and when requested to transmit all of them in a burst. The first option is used while debugging and the second when collecting performance statistics for better accuracy. This system consist three custom software; the first is located in the actual MilCAN Fault Tolerant layer, the second and third is located in the *Data login* computer.

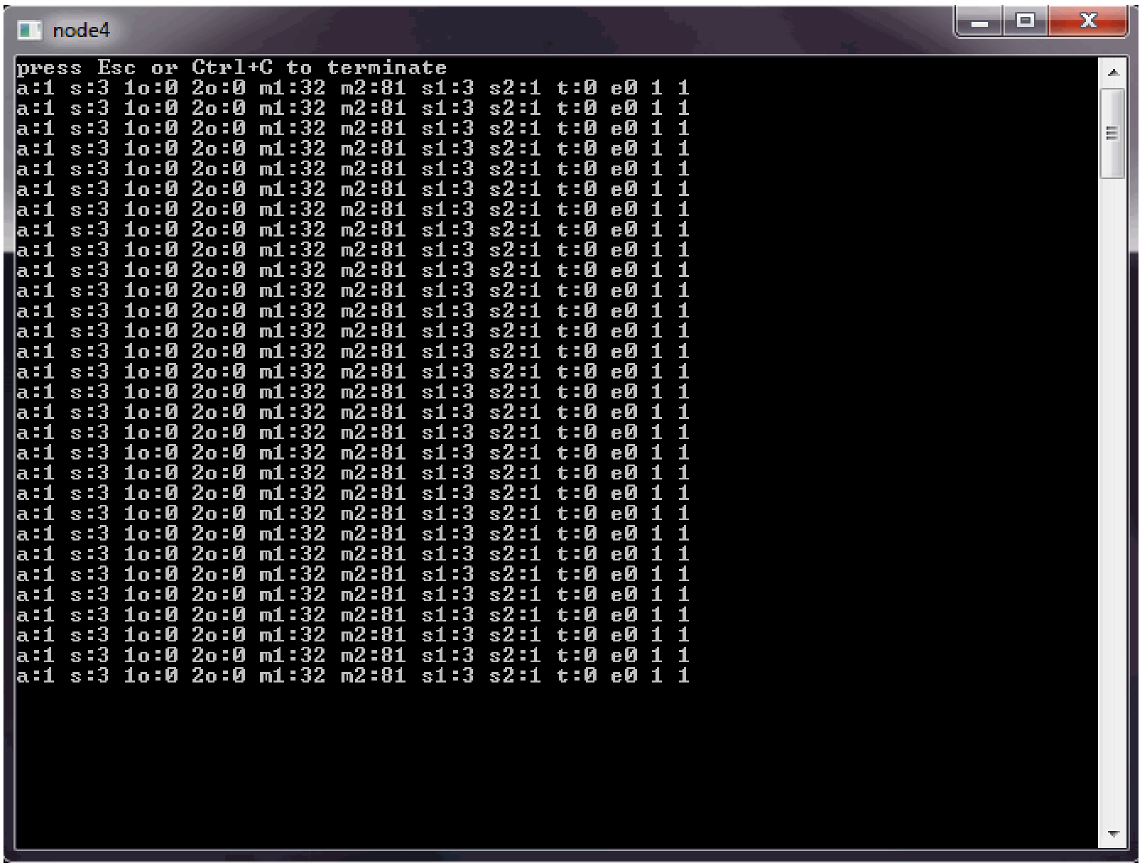
6.4.4.1 Status transmission

The status transmission system is located in the MilCAN FT Layer. On every crucial operation or decision of the FT layer, the status of every important parameter inside the firmware is transmitted or saved in the RAM of the device depending the current configuration. The goal is to use the minimum amount of data and processing power as possible when the data are transmitted over the serial port to the computer. This is achieved by transmitting as less as possible and not to convert the actual data values to the ASCII format. The conversion is done on the computer side were the performance is not an issue. As a result HyperTerminal and other similar software are not capable to capture the incoming data to the computer correctly. For this reason a modified version of a serial terminal program is used.

Additionally in the status transmission system there is also included a performance capturing system, which is used when capturing performance measurements. The performance of the various operations is captured by creating internal performance counters, which are capable to measure various internal operations very accurately. There are four commands for these counters: enable, disable, reset and print. To use these commands in real-time, the node has to receive a predefined CAN message. For this operation CANoe is being used by having each of these messages assigned to be triggered on specific key presses. The **enable** command enables the performance counters, the **disable** command disables the counters, the **reset** command resets the counters to zero and the **print** command sends the captured time measurements to the Serialterm. The reset command helps to synchronise the performance counters across all the connected nodes, where the high priority of the command reassures that it is processed immediately by the node.

6.4.4.2 Serialterm mod

On the **Data login** computer a modified version of the open source software *Serialterm* is used. *Serialterm* is a program developed by Albrecht Schmidt of Lancaster University. It is a command line tool that was developed for debugging microcontroller hardware that is connected to the PC via serial line. The program reads characters from the keyboard and sends them to the selected com-port. At the same time it reads characters from the serial port and displays them on the screen in ASCII, decimal or hexadecimal style. For this software to be used in the current system, some modifications had to be made. The program had to be modified to be able to receive messages from the status transmission system on the devices. Another modification that had to be made was to add the capability for inter-process communication, in order to communicate with the **Status GUI**.



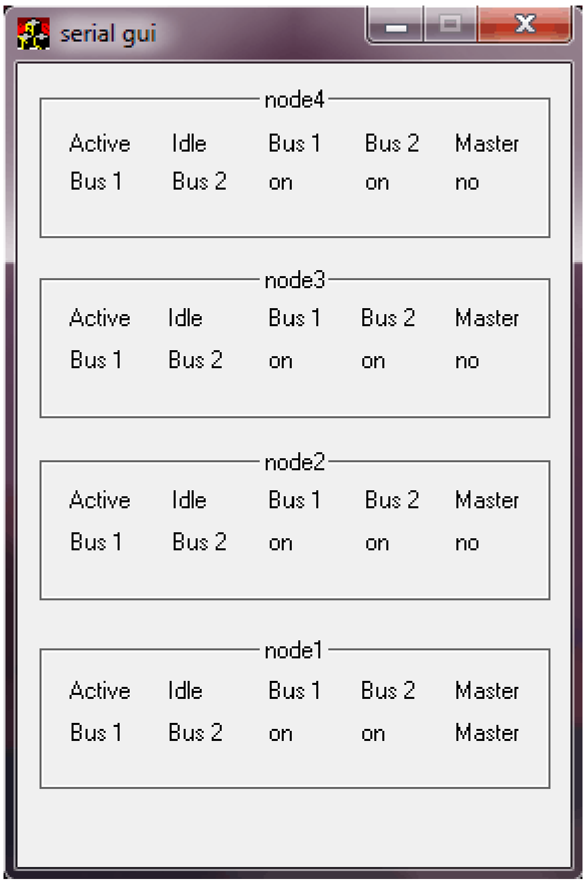
```
press Esc or Ctrl+C to terminate
a:1 s:3 1o:0 2o:0 m1:32 m2:81 s1:3 s2:1 t:0 e0 1 1
a:1 s:3 1o:0 2o:0 m1:32 m2:81 s1:3 s2:1 t:0 e0 1 1
a:1 s:3 1o:0 2o:0 m1:32 m2:81 s1:3 s2:1 t:0 e0 1 1
a:1 s:3 1o:0 2o:0 m1:32 m2:81 s1:3 s2:1 t:0 e0 1 1
a:1 s:3 1o:0 2o:0 m1:32 m2:81 s1:3 s2:1 t:0 e0 1 1
a:1 s:3 1o:0 2o:0 m1:32 m2:81 s1:3 s2:1 t:0 e0 1 1
a:1 s:3 1o:0 2o:0 m1:32 m2:81 s1:3 s2:1 t:0 e0 1 1
a:1 s:3 1o:0 2o:0 m1:32 m2:81 s1:3 s2:1 t:0 e0 1 1
a:1 s:3 1o:0 2o:0 m1:32 m2:81 s1:3 s2:1 t:0 e0 1 1
a:1 s:3 1o:0 2o:0 m1:32 m2:81 s1:3 s2:1 t:0 e0 1 1
a:1 s:3 1o:0 2o:0 m1:32 m2:81 s1:3 s2:1 t:0 e0 1 1
a:1 s:3 1o:0 2o:0 m1:32 m2:81 s1:3 s2:1 t:0 e0 1 1
a:1 s:3 1o:0 2o:0 m1:32 m2:81 s1:3 s2:1 t:0 e0 1 1
a:1 s:3 1o:0 2o:0 m1:32 m2:81 s1:3 s2:1 t:0 e0 1 1
a:1 s:3 1o:0 2o:0 m1:32 m2:81 s1:3 s2:1 t:0 e0 1 1
a:1 s:3 1o:0 2o:0 m1:32 m2:81 s1:3 s2:1 t:0 e0 1 1
a:1 s:3 1o:0 2o:0 m1:32 m2:81 s1:3 s2:1 t:0 e0 1 1
a:1 s:3 1o:0 2o:0 m1:32 m2:81 s1:3 s2:1 t:0 e0 1 1
a:1 s:3 1o:0 2o:0 m1:32 m2:81 s1:3 s2:1 t:0 e0 1 1
a:1 s:3 1o:0 2o:0 m1:32 m2:81 s1:3 s2:1 t:0 e0 1 1
a:1 s:3 1o:0 2o:0 m1:32 m2:81 s1:3 s2:1 t:0 e0 1 1
a:1 s:3 1o:0 2o:0 m1:32 m2:81 s1:3 s2:1 t:0 e0 1 1
a:1 s:3 1o:0 2o:0 m1:32 m2:81 s1:3 s2:1 t:0 e0 1 1
a:1 s:3 1o:0 2o:0 m1:32 m2:81 s1:3 s2:1 t:0 e0 1 1
a:1 s:3 1o:0 2o:0 m1:32 m2:81 s1:3 s2:1 t:0 e0 1 1
a:1 s:3 1o:0 2o:0 m1:32 m2:81 s1:3 s2:1 t:0 e0 1 1
a:1 s:3 1o:0 2o:0 m1:32 m2:81 s1:3 s2:1 t:0 e0 1 1
a:1 s:3 1o:0 2o:0 m1:32 m2:81 s1:3 s2:1 t:0 e0 1 1
a:1 s:3 1o:0 2o:0 m1:32 m2:81 s1:3 s2:1 t:0 e0 1 1
a:1 s:3 1o:0 2o:0 m1:32 m2:81 s1:3 s2:1 t:0 e0 1 1
```

Figure 6.8: Serialterm mod

In figure 6.8 the *Serialterm* mod window is shown displaying raw status data from the node. A new line is added periodically and every time an important action is taken. Some of these information are collected and send to the **Status GUI** which then displays the information in a more user friendly format.

6.4.4.3 Status GUI

The status GUI shows the most important status information for every node on the development testbed. The information on the GUI, are received in real time from the Serialterm mod and also displayed real time.



node4

```
Active   Idle    Bus 1   Bus 2   Master
Bus 1    Bus 2   on      on      no
```

node3

```
Active   Idle    Bus 1   Bus 2   Master
Bus 1    Bus 2   on      on      no
```

node2

```
Active   Idle    Bus 1   Bus 2   Master
Bus 1    Bus 2   on      on      no
```

node1

```
Active   Idle    Bus 1   Bus 2   Master
Bus 1    Bus 2   on      on      Master
```

Figure 6.9: Status GUI

In figure 6.9 the status GUI is displayed. The GUI area is separated in 4 parts that each represents a node. The application shows; which is the active bus and idle bus, the bus status of both buses for each node. It also displays if the node is the FT Master node. The concentrated easy to read information are very important during the development and demonstration of the system allowing the user to have a quick understanding of the status of the system.

6.5 MilCAN Reconfiguration performance

The MilCAN Reconfiguration has been tested on the VSI testbed to verify the correct operation of the added capabilities and to check if the performance of the devices has been affected. To verify the operation of the Reconfiguration, all the functions are tested and verified, by following step by step all the procedures and checking for errors. Since the reconfiguration is not happening during operational mode but in reconfiguration mode, the performance is not important. Performance is important for the operation of the devices that have the reconfiguration capabilities since there are added functions in all the devices and the firmware has been modified by the bootloader, the performance and operation of the devices may have been affected.

To test the reconfiguration capabilities, the devices need to go to configuration mode and then enter the bootloader mode. Since this operation is not meant to happen during operation mode, but during maintenance all the devices enter the configuration mode. From the VSI GUI the targeted device enters the bootloader mode. When it enters the bootloader the status and version is been checked successfully. The status indicates that the targeted node is in configuration mode and in bootloader mode. After that the message configuration, node ID and speed change procedure has been verified successfully according to the response was received from the device. After the completion of all procedures the node was commanded to change back to the application and operational mode. The device then was operating how it was expected according to the new changes. The above procedures were always successful. The flashing of the application firmware was not tested because the VSI GUI was never fully implemented for this operation, although the devices were programmed to support that.

For the evaluation of the performance of the VSI testbed with MilCAN Reconfiguration; various configuration scenarios are followed using part of the testbed. These scenarios include different combination of the MilCAN backbone speeds (250kbit, 500kbit and 1000kbit), MilCAN segments speeds (250kbit, 500kbit and 1000kbit), routing configurations (MilCAN backbone, Ethernet backbone and MilCAN backbone for hard real-time (HRT) & Ethernet backbone for soft real-time (SRT)/ non real-time (NRT)) with different priority distributions (high HRT & low SRT/NRT and low HRT & high SRT/NRT).

The message set of the nodes at the VSI testbed consist of synchronous messages which are transmitted by the nodes with the aim to simulate a drive-by-wire system. These messages, transmitted by each node, are relevant with the operation assigned to them and as a result some are active and some are passive, depending on the circumstances. The role of the active nodes is to transmit and sometimes to receive messages whereas the passive ones only receive. Apart from the messages that are defined by the message set there are also SYNC Frames that assist the synchronisation of the MilCAN. Moreover, the alive messages which are transmitted from all the active nodes that are connected to the bus. Such messages are transmitted with a frequency of 512 sync frames and depending if the bus speed is 1000mbit, 500mbit or 250mbit they have a period of 1.024s, 2.048s or 4.096s respectively.

For simplicity purposes the test layout used for the results presented here is based on two MilCAN segments; the utilities and the automotive segments (figure 6.10). The above mentioned segments are interconnected through a MilCAN and an Ethernet backbone via two gateways. Two VSI Bridges have been used as gateways. The application of routing rules to the VSI Bridges leads to the different routing.

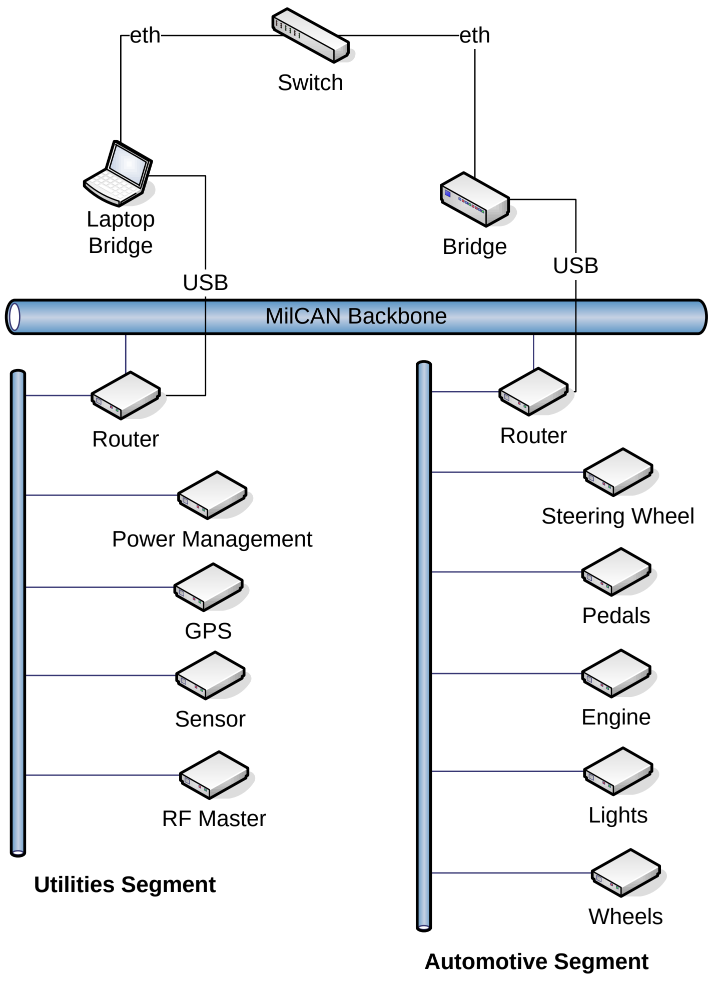


Figure 6.10: Testbed layout

6.5.1 Utilities segment (SEGU)

The utilities segment includes four nodes three of which are active and one passive. The active ones are the GPS node, power management node and the sensor node whereas the passive node is the RF master node. The message set for the active nodes of the utilities segment is demonstrated in table 6.4. Three different frequency configurations exist: a) the normal, b) the high HRT low SRT/NRT, and c) the low HRT high SRT/NRT. The normal is being used during the normal operation of the testbed. Contrarily, the high HRT low SRT/NRT and low HRT high SRT/NRT are for testing scenarios. By modifying the generation frequency of synchronous messages, different traffic profiles are achieved. The period indicated on the tables bellow is the slot period of MilCAN.

Table 6.4: Utilities segment message set

					Normal	**High HRT, low SRT**	**Low HRT, high SRT**
Tag	ID	Prio	Payload	Description		Period (Slots)	
MUGPS1	0x5501	HRT1	8	GPS Position	256	128	512
MUGPS2	0x5502	HRT1	4	GPS Time	256	128	512
MUGPS3	0x5503	HRT1	2	GPS Satellites	256	128	512
MUPWR1	0x5030	SRT1	3	Power load	50	100	25
MUSNS1	0x5936	SRT1	3	X/Y Acceleration	5	10	3
MUSNS2	0x5938	SRT1	3	Vibration	5	10	3

6.5.2 Automotive segment (SEGA)

In the automotive segment consists of five nodes, namely the steering wheel, the pedals, the engine, the wheels and the lights which are all active apart from the lights. The message set for the automotive segment is presented in table 6.5. In accordance with the utilities segment, again, three different frequency configurations exist: a) the normal, b) the high HRT low SRT/NRT, and c) the low HRT high SRT/NRT.

Table 6.5: Automotive segment message set

					Normal	**High HRT, low SRT**	**Low HRT, high SRT**
Tag	ID	Prio	Payload	Description		Period (Slots)	
MAPED1	0x3e30	HRT2	2	Brake	5	3	10
MAPED2	0x3e32	HRT2	2	Throttle	5	3	10
MAPED3	0x5e30	SRT2	2	Brake light	25	50	12
MASTR1	0x3e34	HRT2	2	Steering wheel position	10	5	20
MASTR2	0x3e36	HRT3	2	Gear	10	5	20
MASTR3	0x5830	SRT1	2	Flaps Position	50	100	25
MASTR4	0x5832	SRT1	2	Joystick #2 movement	50	100	25
MASTR5	0x5834	SRT1	2	Joystick #1 movement	50	100	25
MASTR6	0x4430	HRT3	2	Engine status	25	12	50
MAWHE1	0xd220	HRT2	3	Tracks/Tires status	50	25	100
MAWHE2	0xd221	HRT2	3	Tracks/Tires RPM	20	10	40
MAENG1	0xf020	HRT2	2	Engine RPM	10	5	20
MAENG2	0xf021	HRT2	2	Engine temperature	30	15	60
MAENG3	0xf022	HRT3	3	Engine status	30	15	60

6.5.3 Testbed measurements

With the help of CANoe, the traffic dump is collected which later are processed by CANoe Log analyser. Additionally, with the aid of VSI GUI the data was collected from the VSI Bridge. The measurements were approximately 20s long with a 100ms sampling period because of hardware restrictions.

There are six different traffic profiles for every segment and these profiles are the result of a combination of three different MilCAN bus speeds with two different priority distributions. No routing was enabled between the two segments. The characteristics of two of these traffic profiles are presented on table 6.6 till table 6.13. These tables show the periodicity of each individual synchronous MilCAN message transmitted on the bus, including the Sync Frame and Alive messages along with the application messages. The average values are shown to be according the frequency configurations of the message sets for the two traffic setups.

Table 6.6: Automotive MilCAN bus 1000kbit, high HRT low SRT/NRT

Message	ID	Average (ms)	Std. Dev.	Minimum (ms)	Maximum (ms)
MAPED1	0x3e30	6.0974	0.30998	3.45	12.16
MAPED2	0x3e32	6.0974	0.44371	3.18	12.16
MAPED3	0x5e30	104.9	14.454	98.24	202.95
MASTR1	0x3e34	10.191	0.64042	8.33	20.3
MASTR2	0x3e36	10.191	0.68979	7.93	20.55
MASTR3	0x5830	207.76	14.479	201.43	251.91
MASTR4	0x5832	207.76	14.482	201.43	251.77
MASTR5	0x5834	207.76	14.474	201.43	251.77
MASTR6	0x4430	24.446	1.5493	19.4	31.1
MAWHE1	0xd220	51.929	7.4039	48.7	98.9
MAWHE2	0xd221	20.391	1.0379	18.47	40.69
MAENG1	0xf020	10.191	0.6507	8.73	20.07
MAENG2	0xf021	30.558	1.0082	29.03	38.58
MAENG3	0xf022	30.558	1.0346	28.86	38.56

Table 6.7: Utilities MilCAN bus 1000kbit, high HRT low SRT/NRT

Message	ID	Average (ms)	Std. Dev.	Minimum (ms)	Maximum (ms)
MUGPS1	0x5501	259.8	0.36678	258.54	261.24
MUGPS2	0x5502	259.81	0.42403	258.83	261.44
MUGPS3	0x5503	259.81	0.62914	258.44	261.75
MUPWR1	0x5030	217.54	569.93	1.02	1879.01
MUSNS1	0x5936	10.366	1.5177	7.46	20.89

Table 6.8: Automotive MilCAN bus 500kbit, high HRT low SRT/NRT

Message	ID	Average (ms)	Std. Dev.	Minimum (ms)	Maximum (ms)
MAPED1	0x3e30	12.102	0.28938	10.13	16.16
MAPED2	0x3e32	12.102	0.60285	9.86	16.17
MAPED3	0x5e30	206.24	20.991	199.18	298.24
MASTR1	0x3e34	20.228	1.1247	18.89	36.27
MASTR2	0x3e36	20.227	1.1379	18.45	36.15
MASTR3	0x5830	412.68	29.342	402.43	500.05
MASTR4	0x5832	412.68	29.343	402.36	500.05
MASTR5	0x5834	412.68	29.343	402.33	500.05
MASTR6	0x4430	48.547	1.9568	45.22	63.51
MAWHE1	0xd220	103.12	14.967	97.09	197.26
MAWHE2	0xd221	40.454	1.5947	38.87	56.6
MAENG1	0xf020	20.227	1.1525	18.55	36.47
MAENG2	0xf021	60.68	1.9317	59.25	76.58
MAENG3	0xf022	60.681	1.8988	59.2	76.46

Table 6.9: Utilities MilCAN bus 500kbit, high HRT low SRT/NRT

Message	ID	Average (ms)	Std. Dev.	Minimum (ms)	Maximum (ms)
MUGPS1	0x5501	515.8	0.3363	514.81	516.83
MUGPS2	0x5502	515.8	0.35805	514.79	516.86
MUGPS3	0x5503	515.8	1.4554	512.93	518.65
MUPWR1	0x5030	384.19	1071.2	2.81	3727.5
MUSNS1	0x5936	20.226	1.2379	17.26	36.43

Table 6.10: Automotive MilCAN bus 1000kbit, low HRT high SRT/NRT

Message	ID	Average (ms)	Std. Dev.	Minimum (ms)	Maximum (ms)
MAPED1	0x3e30	20.373	0.8054	18.57	28.42
MAPED2	0x3e32	20.373	0.88112	18.51	28.16
MAPED3	0x5e30	24.447	13.456	20.58	30.98
MASTR1	0x3e34	40.827	21.242	38.60	81.22
MASTR2	0x3e36	40.827	2.116	39.04	81.06
MASTR3	0x5830	51.874	71.402	48.51	97.87
MASTR4	0x5832	51.874	71.635	48.30	97.85
MASTR5	0x5834	51.874	71.692	48.30	97.85
MASTR6	0x4430	103.75	10.116	99.12	149.88
MAWHE1	0xd220	207.52	14.017	201.95	251.81
MAWHE2	0xd221	83.336	10.542	80.40	162.68
MAENG1	0xf020	40.827	21.136	39.66	80.96
MAENG2	0xf021	122.98	96.619	121.13	243.65
MAENG3	0xf022	122.98	96.658	120.82	243.63

Table 6.11: Utilities MilCAN bus 1000kbit, low HRT high SRT/NRT

Message	ID	Average (ms)	Std. Dev.	Minimum (ms)	Maximum (ms)
MUGPS1	0x5501	1039.5	0.26349	1038.82	1040.25
MUGPS2	0x5502	1039.5	0.45347	1038.31	1040.76
MUGPS3	0x5503	1039.5	0.54802	1038.17	1040.90
MUPWR1	0x5030	719.66	973.28	7.91	2038.35
MUSNS1	0x5936	10.219	0.84989	8.02	20.33

Table 6.12: Automotive MilCAN bus 500kbit, low HRT high SRT/NRT

Message	ID	Average (ms)	Std. Dev.	Minimum (ms)	Maximum (ms)
MAPED1	0x3e30	40.455	15.734	39.23	56.36
MAPED2	0x3e32	40.454	16.277	39.03	55.67
MAPED3	0x5e30	48.546	19.703	45.78	62.75
MASTR1	0x3e34	80.911	22.203	80.06	97.12
MASTR2	0x3e36	80.911	22.219	79.83	96.93
MASTR3	0x5830	103.12	14.754	98.88	196.20
MASTR4	0x5832	103.12	14.746	98.73	196.05
MASTR5	0x5834	103.12	14.749	98.70	196.04
MASTR6	0x4430	206.24	20.77	199.57	297.67
MAWHE1	0xd220	412.49	28.713	399.51	499.53
MAWHE2	0xd221	164.98	18.816	160.66	257.94
MAENG1	0xf020	80.907	22.603	79.83	97.09
MAENG2	0xf021	242.73	3.855	240.80	258.52
MAENG3	0xf022	242.74	38.429	241.25	258.23

Table 6.13: Utilities MilCAN bus 500kbit, low HRT high SRT/NRT

Message	ID	Average (ms)	Std. Dev.	Minimum (ms)	Maximum (ms)
MUGPS1	0x5501	2063.3	0.12372	2063.08	2063.42
MUGPS2	0x5502	2063.3	0.12372	2063.08	2063.42
MUGPS3	0x5503	2063.3	0.11128	2063.09	2063.37
MUPWR1	0x5030	1186.1	1866.4	15.93	4029.95
MUSNS1	0x5936	20.225	11.672	17.61	36.97

Some messages show a large difference between their average value and their minimum or maximum value. This is happens during the transition between two MilCAN cycles, due to the fact that their assigned periodicity does not overlap with the end of the MilCAN cycle. This affects messages with periods that do not divide clearly the number 1024, as a result the gap between two transmissions to increase or decrease.

6.5.4 Latency Measurements

To measure the latencies the CANoe software is being used, as mentioned above. The message latencies are calculated by monitoring the traffic at each MilCAN bus. At the following tables the summary of these measurements is presented where in the appendix more detailed measurements can be viewed. When routing occurs through the Ethernet backbone and the MilCAN segment speed is 1000kbit the latencies are significantly lower than the other two configurations (see table 6.14). The same applies for the utilities segment at table 6.15. As shown on table 6.16 and table 6.17 although the MilCAN backbone has dropped at 250kbit and the segment’s speed is 1000kbit; the latencies have not been increased. Table 6.18 and table 6.19 verify the previous observations under different speeds.

6.5.4.1 MilCAN backbone 1000kbit, MilCAN segment 1000kbit

Table 6.14: Automotive segment message latencies SEGA to SEGU (ms)

		MilCAN backbone		Ethernet backbone		MilCAN/Ethernet	
ID	Message	high HRT	low HRT	high HRT	low HRT	high HRT	low HRT
0x3e30	MAPED1	0.0556	0.0636	0.0368	0.0323	0.0519	0.0592
0x3e32	MAPED2	0.0569	0.0648	0.0384	0.0359	0.0535	0.0597
0x5e30	MAPED3	0.0723	0.0652	0.0469	0.039	0.0396	0.0298
0x3e34	MASTR1	0.0553	0.0647	0.0365	0.0312	0.0518	0.0623
0x3e36	MASTR2	0.0652	0.0686	0.0465	0.0408	0.0644	0.0646
0x5830	MASTR3	0.0644	0.0712	0.0461	0.0383	0.0344	0.0353
0x5832	MASTR4	0.067	0.0712	0.0471	0.0409	0.0366	0.0358
0x5834	MASTR5	0.0696	0.0713	0.048	0.0427	0.038	0.0357
0x4430	MASTR6	0.0655	0.0661	0.0462	0.0412	0.0647	0.0576
0xd220	MAWHE1	0.0562	0.0651	0.0377	0.0337	0.0534	0.0652
0xd221	MAWHE2	0.035	0.0649	0.0354	0.0321	0.0313	0.0631
0xf020	MAENG1	0.0557	0.0652	0.037	0.0325	0.0522	0.0627
0xf021	MAENG2	0.0566	0.0659	0.0385	0.0342	0.0533	0.0638
0xf022	MAENG3	0.0665	0.0682	0.0482	0.0398	0.0663	0.0645

Table 6.15: Utilities segment message latencies SEGA to SEGU (ms)

		MilCAN backbone		Ethernet backbone		MilCAN/Ethernet	
ID	Message	high HRT	low HRT	high HRT	low HRT	high HRT	low HRT
0x5501	MUGPS1	0.0544	0.071	0.0296	0.0252	0.0672	0.0724
0x5502	MUGPS2	0.0567	0.0709	0.0315	0.0387	0.0681	0.0724
0x5503	MUGPS3	0.0587	0.0709	0.0314	0.0402	0.0685	0.0723
0x5030	MUPWR1	0.0596	0.0659	0.0301	0.0403	0.0329	0.0391
0x5936	MUSNS1	0.0685	0.0603	0.0186	0.0252	0.0305	0.0346

6.5.4.2 MilCAN backbone 250kbit, MilCAN segment 1000kbit

Table 6.16: Automotive segment message latencies SEGA to SEGU (ms)

		MilCAN backbone		MilCAN/Ethernet	
ID	Message	high HRT	low HRT	high HRT	low HRT
0x3e30	MAPED1	0.0552	0.0641	0.0532	0.0603
0x3e32	MAPED2	0.0567	0.0656	0.0548	0.0609
0x5e30	MAPED3	0.0693	0.0653	0.0377	0.0298
0x3e34	MASTR1	0.0549	0.066	0.0529	0.0615
0x3e36	MASTR2	0.0626	0.0688	0.0608	0.0636
0x5830	MASTR3	0.0608	0.0715	0.0335	0.0352
0x5832	MASTR4	0.064	0.0719	0.0344	0.0353
0x5834	MASTR5	0.0687	0.0719	0.0366	0.0352
0x4430	MASTR6	0.0627	0.0672	0.0612	0.065
0xd220	MAWHE1	0.0555	0.0661	0.0543	0.0665
0xd221	MAWHE2	0.0346	0.0663	0.0315	0.063
0xf020	MAENG1	0.0553	0.0663	0.0535	0.0623
0xf021	MAENG2	0.0564	0.0678	0.0549	0.0628
0xf022	MAENG3	0.0638	0.0687	0.0628	0.0638

Table 6.17: Utilities segment message latencies SEGA to SEGU (ms)

		MilCAN backbone		MilCAN/Ethernet	
ID	Message	high HRT	low HRT	high HRT	low HRT
0x5501	MUGPS1	0.0515	0.0686	0.0645	0.0709
0x5502	MUGPS2	0.056	0.0687	0.0656	0.0736
0x5503	MUGPS3	0.0586	0.069	0.0667	0.0749
0x5030	MUPWR1	0.0607	0.0619	0.0297	0.0408
0x5936	MUSNS1	0.0665	0.0593	0.029	0.0359

6.5.4.3 MilCAN backbone 500kbit, MilCAN segment 500kbit

Table 6.18: Automotive segment message latencies SEGA to SEGU (ms)

		MilCAN backbone		Ethernet backbone		MilCAN/Ethernet	
ID	Message	high HRT	low HRT	high HRT	low HRT	high HRT	low HRT
0x3e30	MAPED1	0.0626	0.0431	0.0408	0.0318	0.0607	0.0381
0x3e32	MAPED2	0.0639	0.0453	0.0421	0.0347	0.0612	0.0408
0x5e30	MAPED3	0.0738	0.0389	0.0562	0.03	0.0408	0.0158
0x3e34	MASTR1	0.0654	0.0536	0.0462	0.0401	0.0656	0.0469
0x3e36	MASTR2	0.0699	0.0565	0.0534	0.0396	0.0695	0.0513
0x5830	MASTR3	0.069	0.0501	0.0566	0.0423	0.042	0.0247
0x5832	MASTR4	0.0691	0.0502	0.0578	0.0419	0.0421	0.0256
0x5834	MASTR5	0.0693	0.0505	0.0578	0.0356	0.0422	0.0262
0x4430	MASTR6	0.0681	0.0555	0.0523	0.0412	0.0683	0.0476
0xd220	MAWHE1	0.066	0.06	0.0442	0.043	0.0647	0.0555
0xd221	MAWHE2	0.0576	0.0551	0.0474	0.0421	0.0629	0.0481
0xf020	MAENG1	0.0661	0.0539	0.0465	0.0325	0.0662	0.048
0xf021	MAENG2	0.0684	0.0584	0.049	0.034	0.0691	0.0552
0xf022	MAENG3	0.0721	0.0589	0.0546	0.0398	0.0722	0.0555

Table 6.19: Utilities segment message latencies SEGA to SEGU (ms)

		MilCAN backbone		Ethernet backbone		MilCAN/Ethernet	
ID	Message	high HRT	low HRT	high HRT	low HRT	high HRT	low HRT
0x5501	MUGPS1	0.0667	0.0533	0.0212	0.0197	0.0659	0.057
0x5502	MUGPS2	0.0677	0.0532	0.0211	0.02	0.0662	0.0569
0x5503	MUGPS3	0.0666	0.0504	0.0224	0.0155	0.0657	0.0543
0x5030	MUPWR1	0.0699	0.0487	0.0229	0.0176	0.0305	0.0568
0x5936	MUSNS1	0.0642	0.0358	0.0167	0.0125	0.025	0.0268

6.5.5 Automotive segment

Since the measurements from the Utilities segment have similar characteristics as the ones from the Automotive segment, only the automotive are presented below.

Figure 6.11 and figure 6.12 are generated during the configuration of routing all messages through the MilCAN backbone. As can been seen in figure 6.11 the latencies of all priorities are on the same level although the difference of the bandwidth consumption for the low priority messages is higher. Figure 6.12 clearly demonstrates that higher priority messages have slightly lower latencies.

6.5.5.1 1000kbit MilCAN backbone, 1000kbit MilCAN segment, high HRT and low SRT/NRT

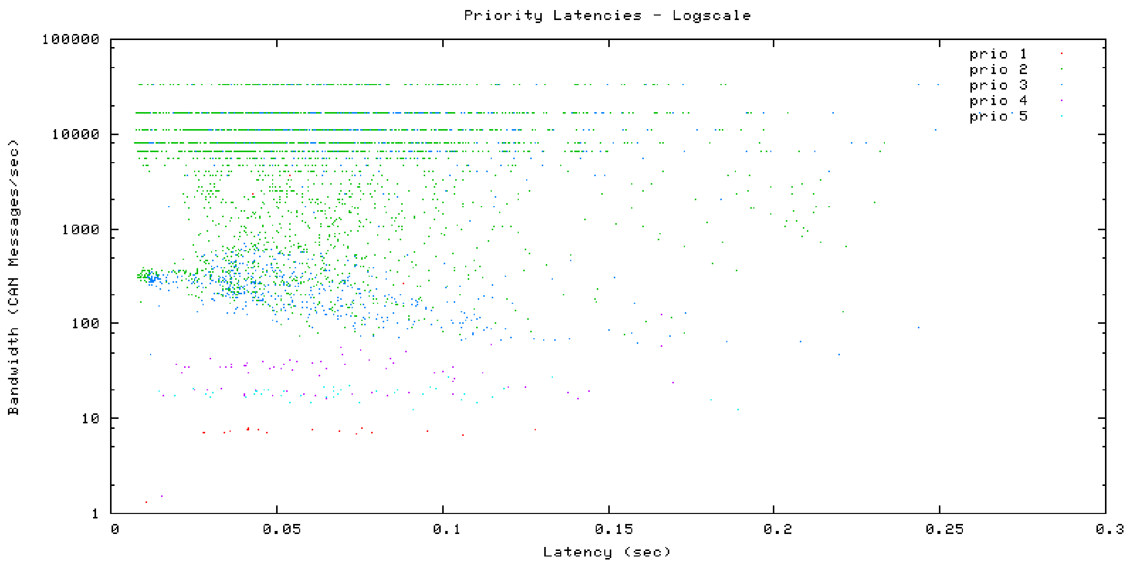


Figure 6.11: SEGA to BBONE message latencies per priority distribution

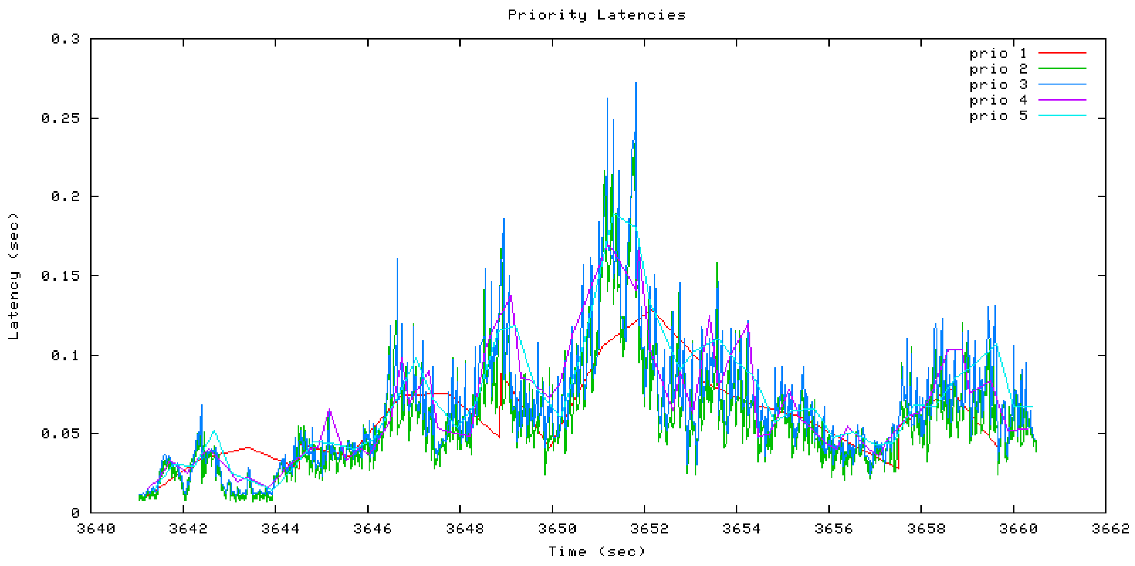


Figure 6.12: SEGA to BBONE message latencies per priority

Figure 6.13 and figure 6.14 are generated during routing all messages through the Ethernet backbone. In figure 6.13 it is visible that the latencies are lower than the previous configuration which also was noticed from the measurements that have been gathered from the CANoe software. In this figure we can also see the bursts of the Ethernet packets. Figure 6.14 shows again that the high priority messages have lower latencies although such a result was not expected due the non-prioritised buffers of TCP/IP layer. This usually occurs because of the high traffic output of the segment which fills the TCP/IP packets quicker.

6.5.5.2 1000kbit MilCAN segment, high HRT and low SRT/NRT

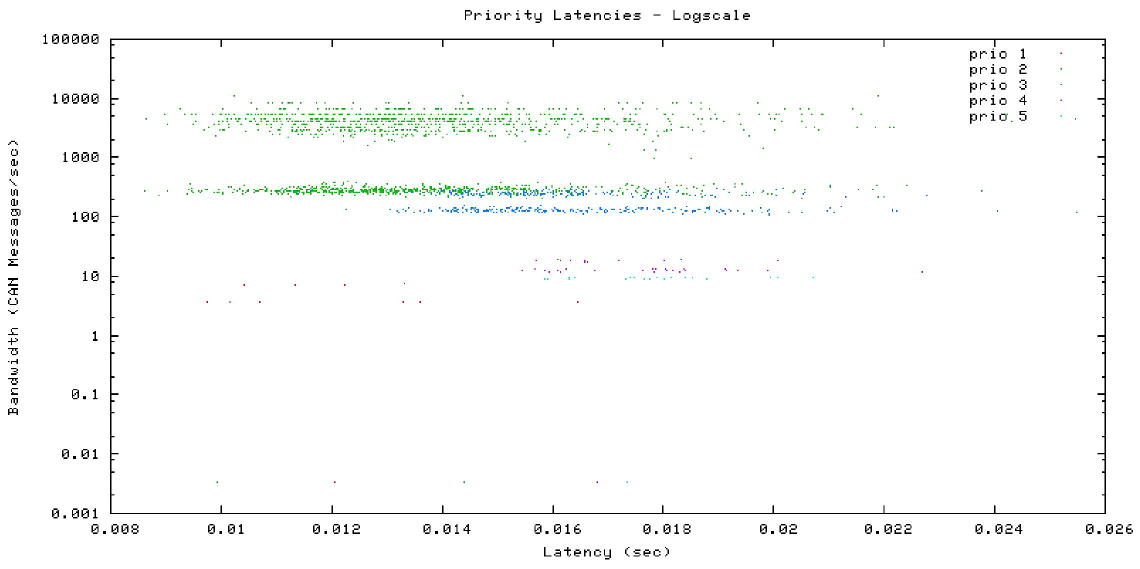


Figure 6.13: SEGA to ETH message latencies per priority distribution

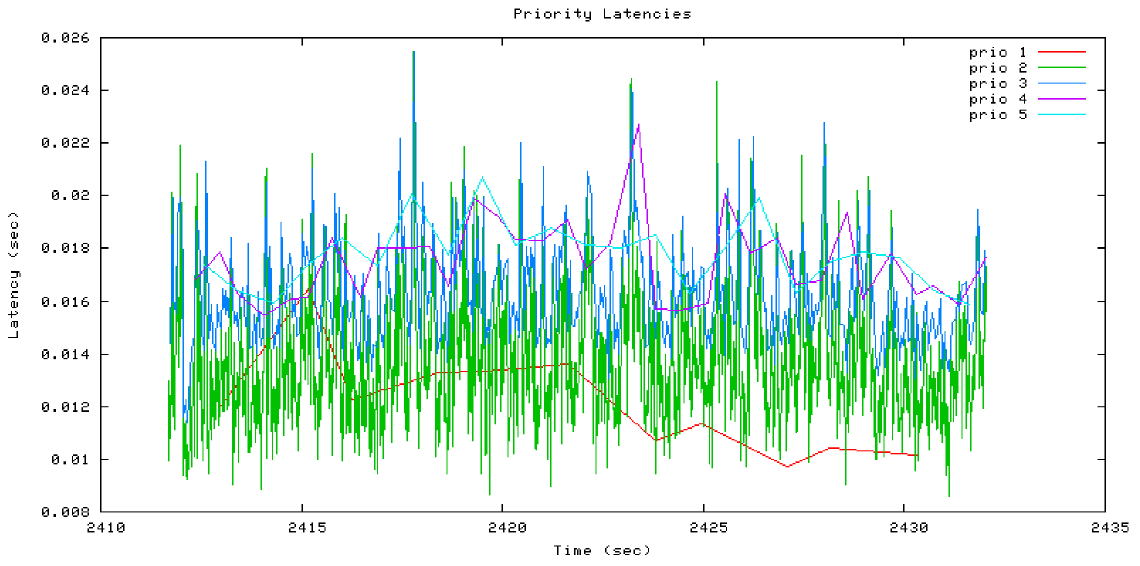


Figure 6.14: SEGA to ETH message latencies per priority

The following configuration combines both backbones, the MilCAN backbone for the HRT messages and the Ethernet for the NRT/SRT messages. In figure 6.15 and figure 6.16 the output through MilCAN is shown and one could easily draw the assumption that high priority messages have lower latencies (see figure 6.16).

6.5.5.3 1000kbit MilCAN backbone, 1000kbit MilCAN segment, high HRT and low SRT/NRT

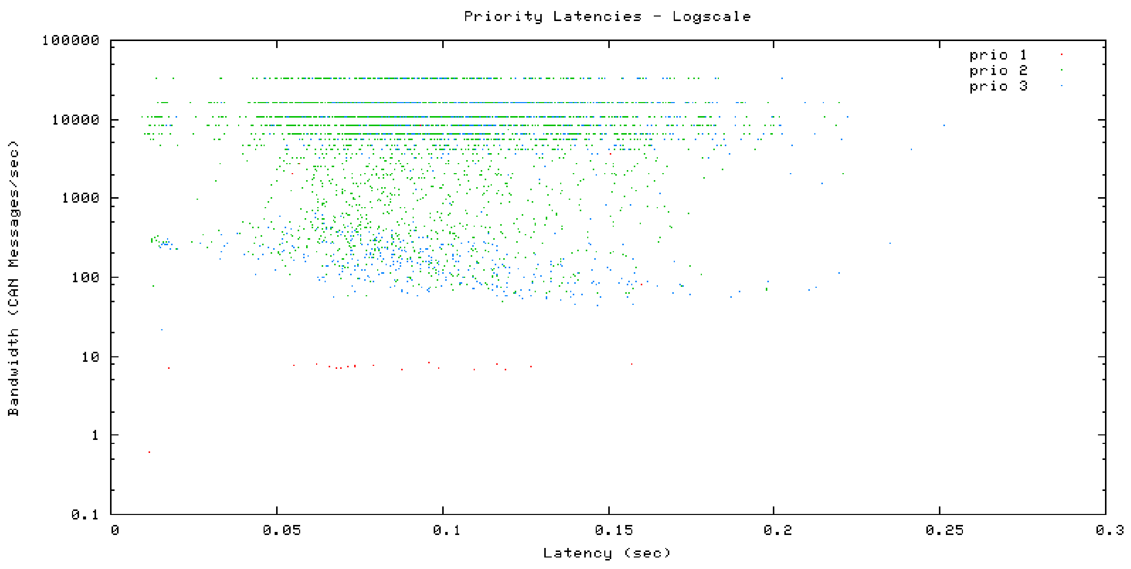


Figure 6.15: SEGA to BBONE message latencies per priority distribution

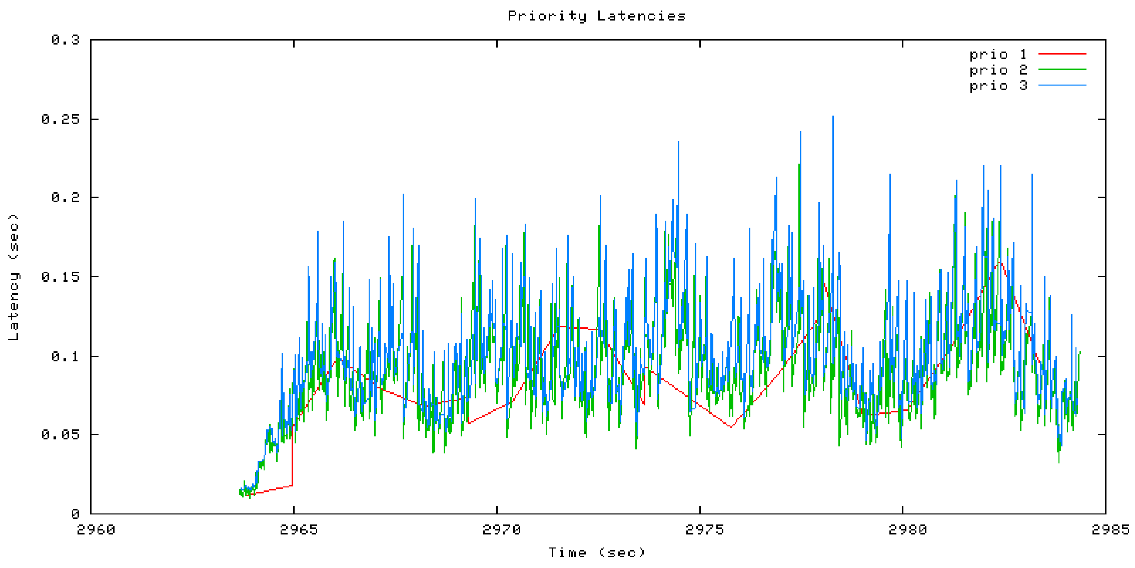


Figure 6.16: SEGA to BBONE message latencies per priority

At figure 6.17 and figure 6.18 the output through the Ethernet backbone is presented. As can been seen in figure 6.18, the higher priority messages do not have lower latencies than the low priority ones. This occurs because of the non-prioritised buffers of TCP/IP layer and although the Ethernet is fast the traffic is not that high to fill the TCP/IP packets quickly enough, which lead us to the conclusion that Ethernet generates low latencies only if the segment's traffic is high. However, it should also be taken into consideration that due to the complexity of the combined routing (more routing commands) the bridge may generate higher latencies because of hardware restrictions.

6.5.5.4 1000kbit MilCAN segment, high HRT and low SRT/NRT

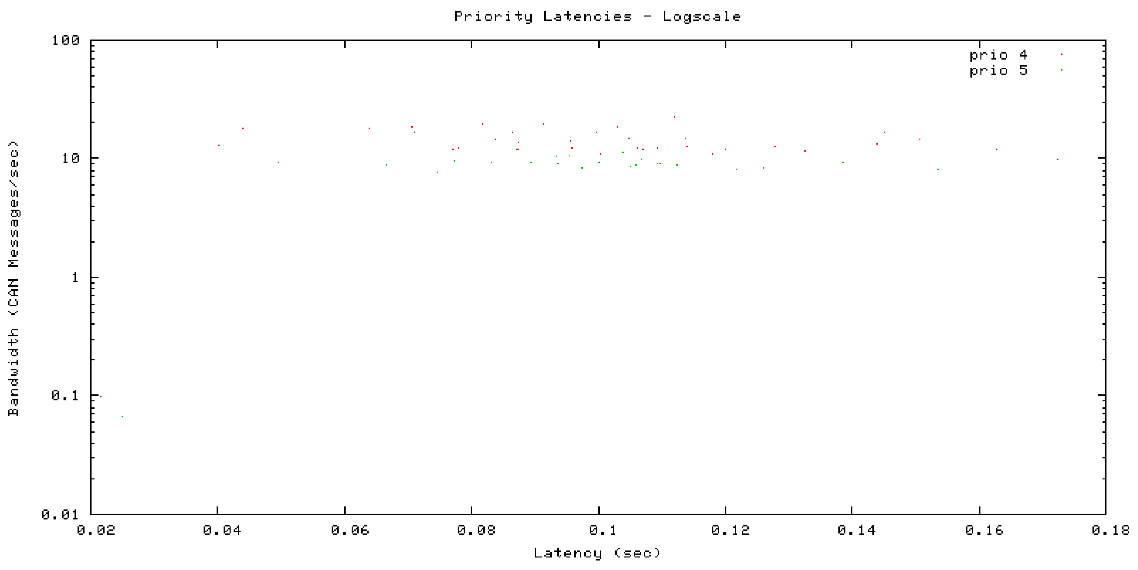


Figure 6.17: SEGA to ETH message latencies per priority distribution

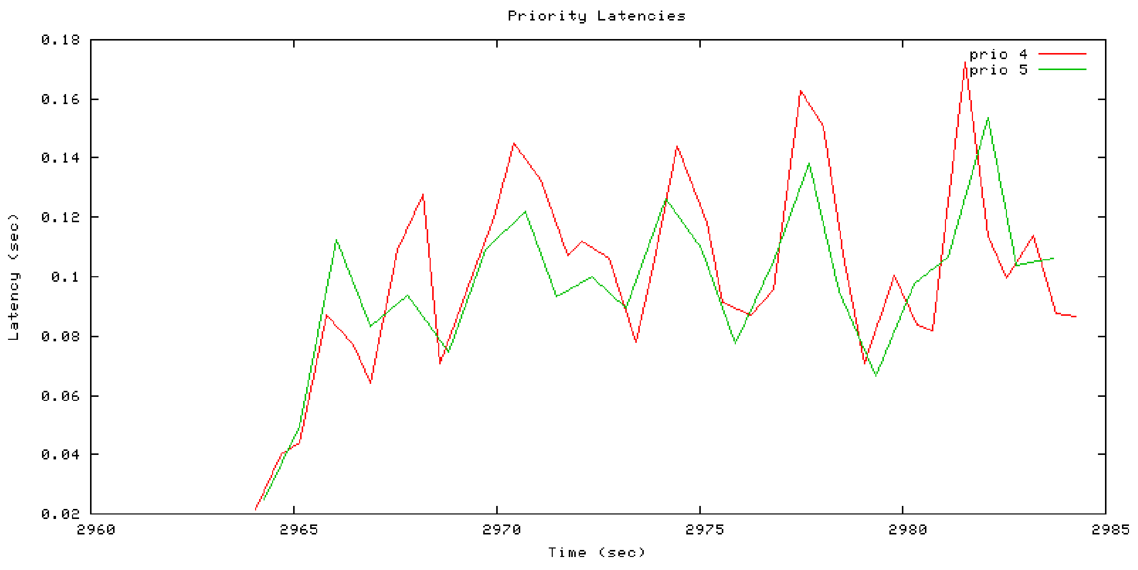


Figure 6.18: SEGA to ETH message latencies per priority

6.6 MilCAN Fault Tolerance performance

The performance of the Fault Tolerant MilCAN has been evaluated through simulations using the internal performance counters. Additionally the application layer that has been used for the tests is specially designed not to affect the operation of the FT layer and provide more accurate measurements. To emulate a normal MilCAN operation it has been configured to generate periodic messages at a MilCAN sync frequency of 5.

The main reason of testing the FT MilCAN layer is to study the reaction of the layer during different fault scenarios. It is important to measure the response and reaction time of the FT layer during these faults. Any missed or delayed scheduled messages should be detected during testing the various fault scenarios. During these tests the nodes have been assigned with an equal weight value. Every fault scenario has been repeated twenty times to acquire more accurate result by calculating their average values.

6.6.1 Non FT Master node loses connection to CAN0

For this scenario a random node loses connection to CAN0 (figure 6.19). When the connection is lost to CAN0, the node detects the non-availability of the bus and informs the FT master. The FT master weights the available buses and decides which bus to use as an active bus.

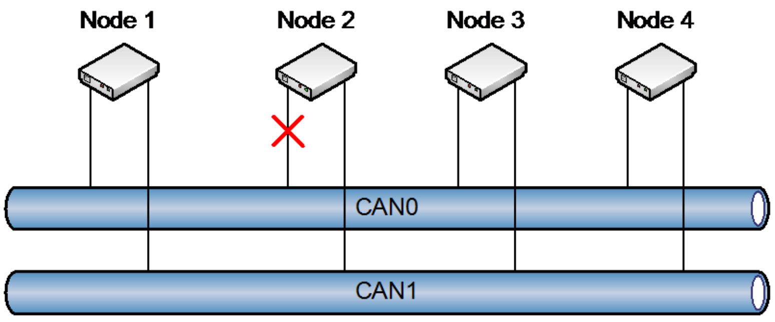


Figure 6.19: Non FT Master node loses connection to CAN0

6.6.1.1 Operation

Before fault, node 1 is the FT master on CAN0 and node 4 on CAN1 where CAN0 is the active bus. When node 2 loses connection to CAN0, reports the fault by sending an asynchronous alive message over CAN1. The FT master nodes (node 1 and node 4), vote according to the WBS algorithm that CAN1 is the new active bus. Node 4 transmits an asynchronous FT master frame which commands the nodes to use CAN1 as active bus.

6.6.1.2 Collected data

During this test, eight time measurements have been taken at the following events:

1. Node 2 detects connection problem with CAN0.
2. Node 2 transmits alive message informing the problem.
3. Node 4 receives alive message.
4. Node 4 votes CAN1 as the new active bus.
5. Node 4 sends FT master frame to all nodes.
6. Node 1 receives the FT master frame and uses CAN1 as active bus.
7. Node 2 receives the FT master frame and uses CAN1 as active bus.
8. Node 3 receives the FT master frame and uses CAN1 as active bus.

Table 6.20: Non FT Master node loses connection to CAN0 collected data (ms)

Event	1	2	3	4	5	6	7	8	9	10
1 → 2	0.28	0.2784	0.4496	0.2784	0.4496	0.2784	0.28	0.2784	0.2784	0.2784
2 → 3	0.9524	1.0656	2.1328	0.3392	3.7168	1.0656	0.832	1.1401	0.8336	2.1328
3 → 4	0.3808	0.3392	0.1696	0.2672	0.3392	0.168	0.3808	0.168	0.3808	0.1696
4 → 5	0.4064	0.9152	0.2784	0.9152	0.2784	0.2784	0.4064	0.2784	0.2768	0.2768
5 → 6	0.9914	1.048	0.6544	0.6544	1.104	1.1047	1.048	0.7648	0.9914	1.255
5 → 7	1.248	1.1886	1.0512	1.0672	1.0512	1.248	0.9536	0.9561	0.6688	1.1886
5 → 8	1.0654	0.798	0.843	0.9761	0.798	0.978	0.8632	0.9671	0.843	1.0654
1 → 8	3.011	3.3964	3.6848	2.4544	5.582	2.7684	2.7624	2.6297	2.4384	3.923

Event	11	12	13	14	15	16	17	18	19	20
1 → 2	0.4069	0.437	0.3382	0.4465	0.3726	0.4314	0.3565	0.3479	0.3049	0.3119
2 → 3	0.6854	0.7162	2.142	1.2144	0.6714	0.3392	0.9441	3.3179	1.0656	0.9524
3 → 4	0.2377	0.3652	0.1733	0.1853	0.179	0.1719	0.2571	0.3808	0.1755	0.2849
4 → 5	0.4251	0.7043	0.815	0.8001	0.4631	0.7653	0.3356	0.9152	0.3837	0.3993
5 → 6	0.7805	0.955	0.9156	0.8391	1.1597	0.8001	1.1342	0.6579	0.7134	0.7321
5 → 7	0.9476	0.7646	0.7961	0.8194	0.9513	0.8363	1.1427	1.2443	1.1737	1.0524
5 → 8	0.9759	0.9789	0.8288	0.8969	0.8064	0.9977	1.0414	0.8945	0.8483	1.0432
1 → 8	2.5356	2.9873	4.2646	3.4657	2.4925	2.5079	2.9347	5.6197	2.6431	2.6806

Table 6.21: Non FT Master node loses connection to CAN0 average values (ms)

Event	Average
1 → 2	0.34417
2 → 3	1.312975
3 → 4	0.258695
4 → 5	0.515855
5 → 6	0.915185
5 → 7	1.017485
5 → 8	0.92546
1 → 8	3.34688

The average time for the whole operation; from detecting the problem till the switch to the new bus is 3.34688ms with minimum 2.4384ms and maximum 5.582ms (table 6.20 & table 6.21). The average time is low and acceptable, where the maximum value can be explained due to processing power restrictions.

6.6.2 FT Master node loses connection to CAN0

For this scenario the FT master node of CAN0 loses connection to CAN0 (figure 6.20). When the connection is lost to CAN0, the node detects the non-availability of the bus and informs the FT master. The FT master weights the available buses and decides which bus to use as an active bus.

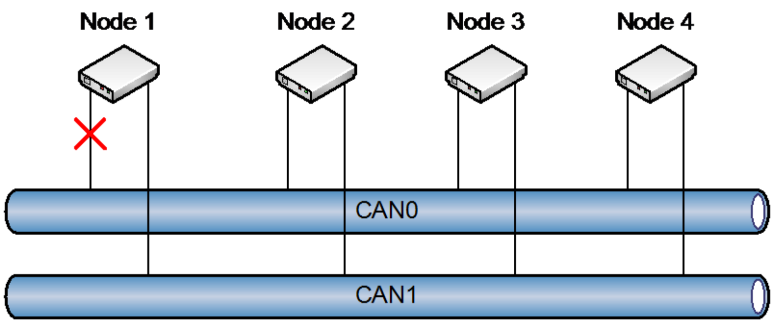


Figure 6.20: FT Master node loses connection to CAN0

6.6.2.1 Operation

Before fault, node 1 is the FT master on CAN0 and node 4 on CAN1 where CAN0 is the active bus. When node 1 loses connection to CAN0, reports the fault by sending an asynchronous alive message over CAN1 and node 2 becomes the FT master of CAN0. The FT master nodes (node 2 and node 4), vote according to the WBS algorithm that CAN1 is the new active bus. Node 4 transmits an asynchronous FT master frame which commands the nodes to use CAN1 as active bus.

6.6.2.2 Collected data

During this test, eight time measurements have been taken at the following events:

1. Node 1 detects connection problem with CAN0.
2. Node 1 transmits alive message informing the problem.
3. Node 4 receives alive message.
4. Node 4 votes CAN1 as the new active bus.
5. Node 4 sends FT master frame to all nodes.
6. Node 1 receives the FT master frame and uses CAN1 as active bus.
7. Node 2 receives the FT master frame and uses CAN1 as active bus.
8. Node 3 receives the FT master frame and uses CAN1 as active bus.

Table 6.22: FT Master node loses connection to CAN0 collected data (ms)

Event	1	2	3	4	5	6	7	8	9	10
1 → 2	0.28	0.1072	0.2784	0.2784	0.2784	0.1072	0.2784	0.28	0.2784	0.3344
2 → 3	0.9808	1.0336	1.2016	0.6976	0.6624	2.9872	0.9808	0.672	1	0.0112
3 → 4	0.2672	0.4592	0.168	0.168	0.1696	0.168	0.1696	0.1696	0.2672	0.1696
4 → 5	0.3744	0.2784	0.2784	0.2784	0.2768	0.2784	0.3744	0.376	0.4896	0.2784
5 → 6	0.6832	1.3728	0.3248	0.8272	0.8624	0.6832	0.7712	0.7664	0.7152	1.512
5 → 7	0.8064	0.5536	0.8064	0.9264	1.6112	0.8368	0.7712	0.8368	0.664	0.9344
5 → 8	0.9056	0.752	1	1.1248	1.5744	0.9056	0.7712	0.8976	0.4752	0.9376
1 → 8	2.5856	2.432	2.2512	2.2496	2.2496	4.224	2.5744	2.264	2.5104	1.728

Event	11	12	13	14	15	16	17	18	19	20
1 → 2	0.299	0.2784	0.1747	0.1228	0.2238	0.1305	0.2528	0.1145	0.2307	0.3091
2 → 3	0.2925	0.1031	0.1312	1.3531	1.7854	0.7393	0.9295	1.5639	0.0967	1.9333
3 → 4	0.2939	0.2199	0.3137	0.4408	0.3711	0.1831	0.2944	0.2749	0.3329	0.4521
4 → 5	0.4751	0.4172	0.3994	0.4057	0.3741	0.33	0.3384	0.4664	0.4809	0.3005
5 → 6	1.0695	0.7916	1.0614	1.1832	0.8828	1.3438	0.7548	1.4712	1.2909	1.0963
5 → 7	0.6592	1.3961	0.8475	1.0752	0.8106	1.2933	1.5998	1.122	1.4012	1.4655
5 → 8	1.3646	1.3539	1.0436	1.4459	0.9096	0.6752	1.4817	1.0615	1.0626	1.4225
1 → 8	2.0197	1.8102	1.8665	3.3976	3.565	2.0581	2.5699	3.4812	2.2038	4.0913

Table 6.23: FT Master node loses connection to CAN0 average values (ms)

Event	Average
1 → 2	0.231855
2 → 3	0.95776
3 → 4	0.26764
4 → 5	0.363545
5 → 6	0.973195
5 → 7	1.02088
5 → 8	1.058255
1 → 8	2.793995

The average time from the time it detects the problem till it switches to the new bus is 2.793995ms with minimum 1.728ms and maximum 4.224ms (table 6.22 table 6.23). The average time is low, where the maximum value can be explained due to processing power restrictions.

6.6.3 Non FT Master node loses connection to CAN1

For this scenario a random node loses connection to CAN1 (figure 6.21). When the connection is lost to CAN1, the node detects the non-availability of the bus and informs the FT master. The FT master weights the available buses and decides which bus to use as an active bus.

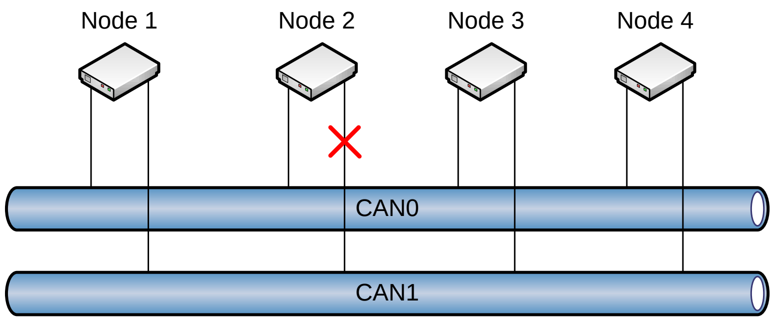


Figure 6.21: Non FT Master node loses connection to CAN1

6.6.3.1 Operation

Before fault, node 1 is the FT master on CAN0 and node 4 on CAN1 where CAN0 is the active bus. When node 2 loses connection to CAN1, reports the fault by sending an asynchronous alive message over CAN0. The FT master nodes (node 1 and node 4), vote according to the WBS algorithm that CAN0 will still remain the active bus. Since there no change nothing is transmitted between the nodes.

6.6.4 FT Master node loses connection to CAN1

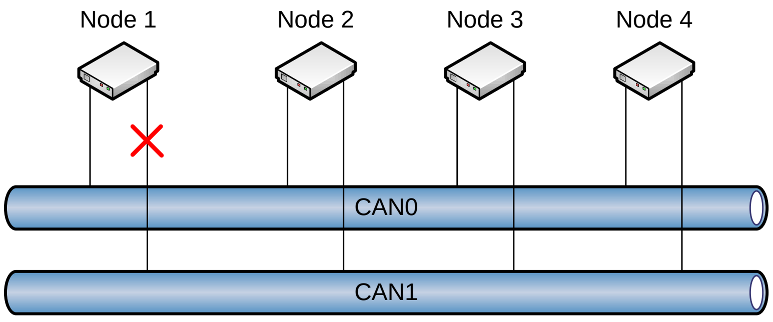


Figure 6.22: FT Master node loses connection to CAN1

For this scenario the FT master node of CAN1 loses connection to CAN1 (figure 6.22). When the connection is lost to CAN1, the node detects the non-availability of the bus and informs the FT master. The FT master weights the available buses and decides which bus to use as an active bus.

6.6.4.1 Operation

Before fault, node 1 is the FT master on CAN0 and node 4 on CAN1 where CAN0 is the active bus. When node 1 loses connection to CAN1, reports the fault by sending an asynchronous alive message over CAN0. The FT master nodes (node 1 and node 4), vote according to the WBS algorithm that CAN0 will still remain the active bus. Since there no change nothing is transmitted between the nodes.

6.6.5 FT Master node loses connection to CAN0 / Non FT Master node loses connection to CAN1

For this scenario the FT master node 1 of CAN0 loses connection to CAN0 and node 2 to CAN1 (figure 6.23). When the connections are lost, the nodes detect the non-availability of the bus and inform the FT master. The FT masters weights the available buses and decide which bus to use as an active bus.

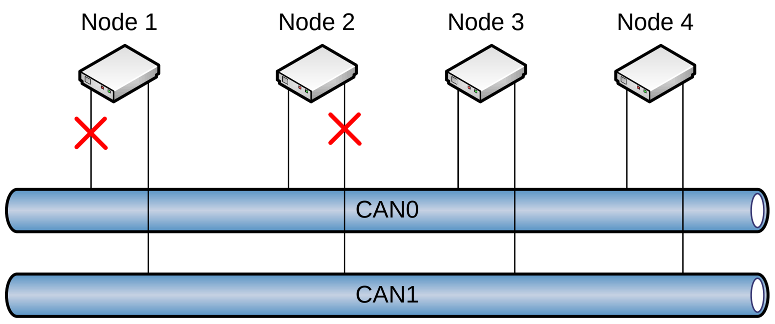


Figure 6.23: FT Master node loses connection to CAN0 / Non FT Master node loses connection to CAN1

6.6.5.1 Operation

Before fault, node 1 is the FT master on CAN0 and node 4 on CAN1 where CAN0 is the active bus. When node 1 loses connection to CAN0, reports the fault by sending an asynchronous alive message over CAN1 and node 2 becomes the FT master of CAN0. Also when node 2 loses connection to CAN1, reports the fault by sending an asynchronous alive message over CAN0. The FT master nodes (node 2 and node 4), vote according to the WBS algorithm that CAN0 will still remain the active bus since the weight on both buses stays the same. Since there no change nothing is transmitted between the nodes.

6.6.6 CAN0 split in half

For this scenario CAN0 is been split at half (figure 6.24). The nodes detect the non-availability of the bus and inform the FT master. The FT masters weights the available buses and decide which to use as active bus.

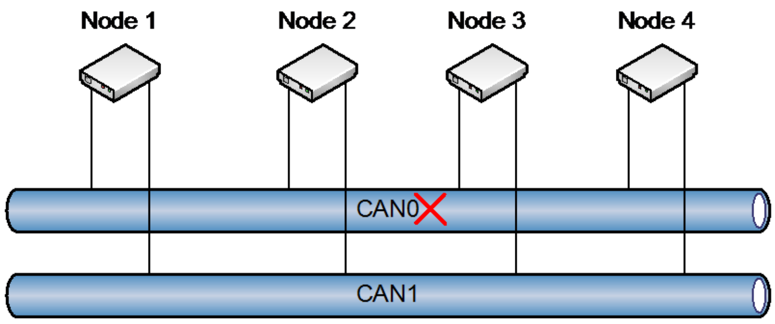


Figure 6.24: CAN0 segmented in half

6.6.6.1 Operation

Before fault, node 1 is the FT master on CAN0 and node 4 on CAN1 where CAN0 is the active bus. When CAN0 is been split at half, node 1 continue to be the FT master on 1st half of CAN0 and node 3 becomes the FT master on 2nd half of CAN0. Node 4 remains to be the FT master of CAN1. The FT master nodes (node 1, node 3 and node 4), vote according to the WBS algorithm that CAN1 is the new active bus. Node 4 transmits an asynchronous FT master frame which commands the nodes to use CAN1 as active bus.

6.6.6.2 Collected data

During this test, eight time measurements have been taken at the following events:

1. Node 3 becomes new master on 2nd half of CAN0.
2. Node 3 transmits the first FT master frame on CAN0.
3. Node 4 detects that node 3 is the new master on CAN0.
4. Node 4 votes CAN1 as the new active bus.
5. Node 4 sends FT master frame to all nodes.
6. Node 1 receives the FT master frame and uses CAN1 as active bus.
7. Node 2 receives the FT master frame and uses CAN1 as active bus.
8. Node 3 receives the FT master frame and uses CAN1 as active bus.

Table 6.24: CAN0 segmented in half collected data (ms)

Event	1	2	3	4	5	6	7	8	9	10
1 → 2	0.4464	0.5088	0.3472	0.448	0.2784	0.5088	0.3744	0.376	0.4816	0.4464
2 → 3	1.3744	1.9872	2.7008	1.3744	1.7696	1.6976	1.9872	1.2496	2.16	2.928
3 → 4	0.0592	0.0592	0.0592	0.0592	0.0592	0.0592	0.0592	0.0592	0.0592	0.0592
4 → 5	0.1056	0.2768	0.2752	0.2768	0.2752	0.1056	0.1056	0.104	0.4192	0.1056
5 → 6	1.7696	2.3488	1.0416	0.8624	2.2528	2.3488	3.1472	1.808	1.7696	1.9776
5 → 7	2.7808	2.0032	0.9976	0.8848	1.368	2.0032	2.7808	1.792	1.8656	1.6736
5 → 8	1.6048	2.1776	1.6048	1.1696	2.1776	2.4928	3.1888	2.2608	1.0544	1.2976
1 → 8	3.5904	4.8352	4.38	3.0208	3.7504	4.3744	5.3072	3.5808	4.1744	4.8368

Event	11	12	13	14	15	16	17	18	19	20
1 → 2	0.4402	0.2934	0.448	0.4305	0.4174	0.4057	0.2785	0.3472	0.4616	0.444
2 → 3	1.7478	2.8461	2.774	1.416	2.7736	2.8231	1.9185	1.2814	2.4338	1.3089
3 → 4	0.0592	0.0592	0.0592	0.0592	0.0592	0.0592	0.0592	0.0592	0.0592	0.0592
4 → 5	0.1056	0.2087	0.1329	0.249	0.3879	0.2845	0.2693	0.1056	0.1679	0.2986
5 → 6	2.1472	0.8765	2.2483	1.6715	1.147	1.0194	0.8742	1.0654	1.5728	2.2525
5 → 7	1.121	1.0289	2.1454	2.1573	0.9465	2.3036	1.8761	1.3958	2.4401	1.8078
5 → 8	1.9945	2.1679	2.4209	1.1782	1.3122	2.2707	3.1167	2.5205	1.6878	1.6409
1 → 8	3.4738	4.2839	5.5595	3.3329	4.5846	4.5919	3.3997	2.8588	4.6953	3.7516

The average time for the whole operation, from the re-election of new FT master till the switch to the new bus is 4.418965ms with minimum 3.0208 and maximum 5.5595ms (table 6.24 table 6.25). The average time is low, where the maximum value can be explained due to processing power restrictions.

Table 6.25: CAN0 segmented in half average values (ms)

Event	Average
1 → 2	0.409125
2 → 3	2.0276
3 → 4	0.0592
4 → 5	0.21298
5 → 6	1.71006
5 → 7	1.768605
5 → 8	1.966955
1 → 8	4.418965

6.6.7 CAN1 split in half

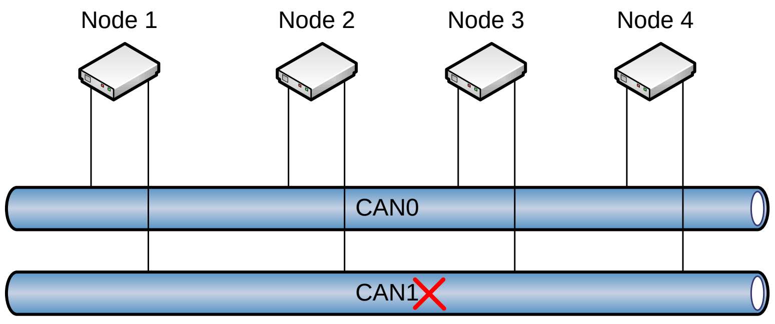


Figure 6.25: CAN1 split in half

For this scenario CAN1 is been split at half (figure 6.25). The nodes detect the non-availability of the bus and inform the FT master. The FT masters weight the available buses and decide which bus to use as an active bus.

6.6.7.1 Operation

Before fault, node 1 is the FT master on CAN0 and node 4 on CAN1 where CAN0 is the active bus. When CAN1 is been split at half, node 4 continue to be the FT master on 2nd half of CAN1 and node 2 becomes the FT master on 1st half of CAN1. Node 1 remains to be the FT master of CAN0. The FT master nodes (node 1, node 2 and node 4), vote according to the WBS algorithm that CAN0 will remain the active bus.

6.6.8 CAN0 increased capacitance

With the help of CANstress it is possible to increase the capacitance between the CAN high and CAN low of any CAN bus. This will result the affected bus to be virtually longer. According to the CAN specifications different operational speeds require different bus lengths to ensure correct operation. By increasing the capacitance between the CAN high and low of CAN0 will render the bus in-operational. The FT master will decide which will be the operational speed for the bus.

6.6.8.1 Operation

Before fault, node 1 is the FT master on CAN0 and node 4 on CAN1 where CAN0 is the active bus. The operational speed of CAN0 is 1000kbit and for CAN1 is 500kbit. When the capacitance in CAN0 is increased the bus becomes unusable. When all the nodes connected on the bus detect the problem, report the fault by sending an asynchronous alive message over CAN1. Since the bus cannot be repaired by resetting the bus, the CAN0 FT master node (node 1) decides according to the error recovery algorithm to drop the speed to 500kbit. Node 1 transmits an asynchronous FT master frame over CAN1 which commands the nodes to use 500kbit as an operational speed for CAN0. All the nodes configured themselves to use 500kbit as the operational speed for CAN0 and it becomes again operational.

6.6.9 CAN0 error injection

With the help of CANstress it is possible to introduce illegal frames in to the CAN bus. For this test the CRC field of the CAN frame will be corrupted and it will not match with the content of the frame. According to the CAN specifications this will be detected as an error frame. The FT masters weights the available buses and decide which to use as active bus.

6.6.9.1 Operation

Before fault, node 1 is the FT master on CAN0 and node 4 on CAN1 where CAN0 is the active bus. When the error frames are injected in to CAN0, the nodes report the fault by sending an asynchronous alive message over CAN1. The FT master nodes (node 1 and node 4), vote according to the WBS algorithm that CAN1 is the new active bus. Node 4 transmits an asynchronous FT master frame which commands the nodes to use CAN1 as active bus. When the error injection stops, CAN0 becomes operational. The FT master nodes (node 1 and node 4), vote according to the WBS algorithm that CAN0 is the new active bus. Node 1 transmits an asynchronous FT master frame which commands the nodes to use CAN0 as active bus.

6.7 Conclusion

The use of two testbeds provides the capability to have the High Availability MilCAN tested on a variety of conditions. The development testbed provides the ability to test the operation in an isolated environment and verify the operation in detail, where the VSI testbed gives ability to test the High Availability MilCAN in a real life vehicle environment.

The performance of the High Availability MilCAN has been tested on the development testbed and the VSI testbed using internal performance mechanisms and external monitoring tools to generate the latencies throughout of the system. The VSI testbed provides the interconnection of the MilCAN segments through various combinations of MilCAN and Ethernet backbones. For the specific tests three combinations were followed: either routing all messages through MilCAN or Ethernet backbone or the HRT through MilCAN or SRT/NRT through Ethernet. The measurements taken are an indication of the performance of the system.

By using non-deterministic backbones such as Ethernet, the performance of the system is affected as expected. The results from the various test scenarios and traffic policies have proven close to the theoretical expectations. This proves that the High Availability MilCAN has not affected the performance of the system.

The close observation of the operation of the Fault Tolerant capabilities under different fault scenarios has also proven to be what was expected. The reaction times for the measured operations were acceptable with only a few exceptions that were longer. The reason for these exceptions is processing power restrictions; causing delays during the operation.

Chapter 7 Conclusion and future work

7.1 Conclusion

The work presented in this thesis investigates how to add two extra functionalities to MilCAN, an already existing real-time communication protocol. Two testbeds are designed and developed to evaluate the concept and provide an initial feedback for further development.

Throughout the study of fieldbuses used in the industry, the basic principles for the design of the system were identified. The MilCAN protocol is investigated in more detail to identify any limitations to address in this research. After completing the background research, the design of the MilCAN Reconfiguration is provided followed by a detailed explanation of its components and operation. Additionally the MilCAN Fault Tolerance is then provided followed again by a detailed explanation of its operation. Concluding this thesis, the two testbeds that were developed for evaluating the two MilCAN additions are presented, leading to performance results and operation verification.

7.1.1 MilCAN Reconfiguration

The MilCAN Reconfiguration addition was designed to support the Through Life Capability Management that is not currently available in the MilCAN protocol. This addition is beneficial to the engineers while maintaining MilCAN devices by providing the functionality to access and configure any device remotely. The reconfiguration capabilities cover the software side of the maintenance like configuring the message set and upgrading the firmware. The devices are accessed remotely which allows all the operations to be completed from a central location.

To enable the MilCAN Reconfiguration capabilities a set of additional design requirements are added to the already existing MilCAN protocol. The design consist of three parts; the bootloader, the reconfiguration protocol and the VSI GUI. The bootloader offers the local operations on every node, by utilising various techniques to achieve normal operation next to the application layer. The reconfiguration protocol is used to communicate remotely to the bootloader and supports all the reconfiguration operations. The VSI GUI provides the user interface; to control all the bootloader's operations through the custom communication protocol.

During the development of the MilCAN Reconfiguration many lessons were learned from mistakes during the design and implementation period. The development of the bootloader was particularly difficult because it had to coexist with the application layer. The system was tested on the VSI testbed, where its operation was verified along with performance measurements. The performance of the MilCAN Reconfiguration enabled node is analysed and detailed performance graphs have been presented. The results that were gathered, prove that the MilCAN Reconfiguration has not affected the operation and performance of the testbed. More specifically, although all the priorities have the same latencies, higher priority messages have slightly lower latencies. Additionally it was observed that the Ethernet generates low latencies only if the segment's traffic is high.

7.1.2 MilCAN Fault Tolerance Layer

The MilCAN Fault Tolerance Layer is meant to provide the additional fault tolerance capability that is currently missing from the MilCAN protocol in order to enhance the network availability. With the use of the FT Layer a longer continuity of service can be provided by overcoming any fault and errors introduced in the network.

To provide the MilCAN FT capabilities the Fault Tolerant requirements are added to the existing MilCAN protocol. Since it is a software solution it is flexible to be included to existing MilCAN devices with no modification to the physical layer and additionally be hardware independent. The design of the layer is divided in three different blocks depend the system configuration. The three blocks are the error detection, bus switching and error recovery. The bus switching is only enabled on devices that have more than one bus available. The FT layer is provides the ability to transparently inter-connect the application layer to multiple buses using a common virtual interface.

During the development of the MilCAN Fault Tolerance layer many lessons were learned from mistakes during the design and implementation period. The implementations helped to identify various weaknesses on the initial designs and provided the feedback required to lead to a better design. The synchronised operation of the FT enabled nodes has been proven particularly difficult requiring a fool proof communication between them. The layer has been tested on the development testbed to verify the operation under different error scenarios. The results gathered verified the operation of the FT layer.

7.2 Future work

Further work for the High Availability MilCAN will be to be implemented on a mobile demonstrator that is an actual operational vehicle. The mobile demonstrator is a commercial of the shelf Buggy vehicle that has been used as the base of various implementations for the Vetronics Research Centre. The author has already installed a MilCAN single bus network to control a set of on-board devices. MilCAN provides full control over the engine and lights, additionally it collects information from the on board sensors speed and engine temperature.

The controls are received from a GUI located on a touch screen which is installed on the vehicle. The single bus implementation has been proven difficult because of space restrictions and the operational conditions of the vehicle. The space restriction affects the placement of the required devices and the installation of the bus. The operational conditions are vibrations and external interferences which have been proven to be a lot different compared to the previously constructed testbeds. In order to test and verify the operation of the High Availability MilCAN on the Buggy, a second CAN bus needs to be installed in order to operate MilCAN in dual bus mode. This will provide us a full functional system in an actual vehicle.

The underlying CANbus protocol has a failure mode called the babbling idiot, whereby a fault node can gain high priority access to the bus. Although this fault is not common, it would potentially jeopardise the sync frame broadcast and hence the determinism of the message delivery schedule of the system. The babbling idiot problem has been addressed in a theoretical level during this research, but since it is not a common fault it is hard to emulate. Further research is required in order to be able to emulate accurately the failure and evaluate the operation of the theoretical proposed solution.

Additionally, it will be useful if MilCAN provides Health and Usage Monitoring Systems (HUMS) functionalities. HUMS increase the availability of systems whilst reducing the maintenance. MilCAN already provides some health monitoring functionalities by being able to detect bus errors. Since High Availability MilCAN has introduced new components, their operation needs to be monitored and recorded. Furthermore MilCAN should be able to provide Build In Test (BIT) capabilities. This requires Built In Test Equipment (BITE) to be used with MilCAN devices. Additionally further investigation is required to provide additional health monitoring capabilities by monitoring the FT layer. More specifically there have to be counters that provide information such as; how often the bus goes to bus-off, how long is non-operational, how often traffic is switched and how often the speed has dropped.

References

[Charchalakis'05] Periklis Charchalakis, "Integrated vetronic systems - Intelligent bridging of vehicle networks over high speed backbones," PhD, Engineering and Design, University of Sussex, Brighton, 2005.

[Valsamakis'06] George Valsamakis, "Vetronic Systems Integration: Network Management and (Re)Configurability Integration on MilCAN based systems," PhD, Engineering and Design, University of Sussex, Brighton, 2006.

[Pinceti'04] P. Pinceti, "Fieldbus: more than a communication link," *Instrumentation & Measurement Magazine, IEEE,* vol. 7, pp. 17-23, 2004.

[Pierre Thomesse'99] Jean Pierre Thomesse, "Fieldbuses and interoperability," *Control Engineering Practice,* vol. 7, pp. 81-94, 1999.

[Glanzer'96] David A. Glanzer and Charles A. Cianfrani, "Interoperable fieldbus devices: a technical overview," *ISA Transactions,* vol. 35, pp. 147-151, 1996.

[Patzke'98] Robert Patzke, "Fieldbus basics," *Computer Standards & Interfaces,* vol. 19, pp. 275-293, 1998.

[Schumny'98a] Harald Schumny, "Fieldbuses in measurement and control," *Computer Standards & Interfaces,* vol. 19, pp. 295-304, 1998a.

[Zimmermann'80] H. Zimmermann, "OSI Reference Model--The ISO Model of Architecture for Open Systems Interconnection," *Communications, IEEE Transactions on,* vol. 28, pp. 425-432, 1980.

[Powers'00] William F. Powers and Paul R. Nicastri, "Automotive vehicle control challenges in the 21st century," *Control Engineering Practice,* vol. 8, pp. 605-618, 2000.

[Leen'02] G. Leen and D. Heffernan, "Expanding automotive electronic systems," *Computer,* vol. 35, pp. 88-93, 2002.

[TUW'97] TUWVienna University of Technology. (2004, 28/12). *TTP/C*. Available: http://www.vmars.tuwien.ac.at/projects/ttp/ttpc.html

[Bannatyne'98] Ross Bannatyne, "Building fault tolerant embedded systems using TTP/C," *Electronics Engineer,* Nov. 1999 1998.

[Kopetz'01] Hermann Kopetz, "A Comparison of TTP/C and FlexRay," ed, 2001.

[Consortium'04] FlexRay Consortium, "FlexRay Communications Systems Protocol Specification v2.0," ed, 2004.

[FlexRay'05] FlexRayFlexRay Consortium. (2004, 28/12). *FlexRay basics*. Available: http://www.flexray.com/about.php?menuID=1

[FlexRay'04] FlexRayIXXAT. (2005, 4/4). *FlexRay Introduction*. Available: http://www.ixxat.com/introduction_flexray_en.html

[ODVA'04] ODVA. *Open DeviceNet Vendor Association*. Available: http://www.odva.org/

[Hitex'95] Hitex. (1995, January 1995) Hitex UK Ltd. Controller Area Networking: The Future of inductrial Microprocessor Communications. *Hitex UK Ltd C51/166 newsletter*.

[softing'05] softing. *CAN bus*. Available: http://www.softing.com/home/en/industrial-automation/products/can-bus/index.php?navanchor=3010024

[ISO'93]ISO, "ISO 11898," in *Road vehicles - Interchange of digital information- Controller area network (CAN) for high-speed communication*, ed: International Organization for Standardization, 1993.

[MWG'03a] MilCAN Working Group MWG, "MilCAN A Physical Layer Specification IHSDB-APP-GEN-D-030 Revision 3," ed: MilCAN Working Group, 2003a.

[Group'03] MilCAN Working Group, "MilCAN A Data Link Layer Specification IHSDB-APP-GEN-D-031 Revision 4," ed: MilCAN Working Group, 2003.

[MWG'03b] MilCAN Working Group MWG, "MilCAN A Application Layer specification IHSDB-APP-GEN-D-032 Revision 2," ed: MilCAN Working Group, 2003b.

[MWG'09] MilCAN Working Group MWG, "MilCAN A Specification MWG-MILA-001 Revision 3," ed: MilCAN Working Group, 2009.

[ARM'04] ARM. (2010, 26 May). *Instruction Set*. Available: http://www.keil.com/support/man/docs/a166/a166_xa.htm

[ARM'05] ARM. (2010, 26 May). *VECTAB Linker Directive*. Available: http://www.keil.com/support/man/docs/l166/l166_vectab.htm

[Wikipedia'10a] WikipediaWikipedia, The Free Encyclopedia. (2010, 6 May). *Intel HEX*. Available: http://en.wikipedia.org/w/index.php?title=Intel_HEX&oldid=360347177

[Infineon'00] Infineon. (2000). *C167CS Derivatives - 16-Bit Single-Chip Microcontroller*.

[Wikipedia'10b] WikipediaWikipedia, The Free Encyclopedia. *Failure mode and effects analysis*. Available: http://en.wikipedia.org/w/index.php?title=Failure_mode_and_effects_analysis&oldid=387361911

[Temple'98] C. Temple, "Avoiding the babbling-idiot failure in a time-triggered communication system," in *Fault-Tolerant Computing, 1998. Digest of Papers. Twenty-Eighth Annual International Symposium on*, 1998, pp. 218-227.

[Vector'06a] Vector. (2006, May). *Test with CANoe*. Available: http://www.vector.com/vi_canoe_en.html

[Vector'06b] Vector. (2006, May). *CANstressD and CANstressDR*. Available: http://www.vector.com/vi_canstress_en.html

[Johnson'88] B. W. Johnson, Design & analysis of fault tolerant digital systems. Boston: Addison-Wesley Longman Publishing Co., 1988.

[Oikonomidis'08] Panagiotis Ioannis Oikonomidis, Elias Stipidis, Periklis Charchalakis, et al., "MilCAN Fault Tolerance Layer," presented at the 12th International CAN conference, Barcelona Spain, 2008.

[Oikonomidis'09] Panagiotis Ioannis Oikonomidis, "MilCAN Fault Tolerance Layer Specifications," presented at the 28th MilCAN Working Group Briefing, Brighton UK, 2009.

[Stallings'06] William Stallings, "Cryptography and network security principles and practices," ed. Princeton, N.J.: Recording for the Blind & Dyslexic, 2006.

[Iee'90] IEEE, "IEEE Standard Glossary of Software Engineering Terminology," 1990.

[Isermann'02] R. Isermann, R. Schwarz. and S. Stolzl, "Fault-tolerant drive-by-wire systems," *Control Systems Magazine, IEEE,* vol. 22, pp. 64-81, 2002.

[Fredriksson'02] L. B. Fredriksson, "CAN for critical embedded automotive networks," *Micro, IEEE,* vol. 22, pp. 28-35, 2002.

[Hammett'03] R. Hammett, "Flight-critical distributed systems: design considerations [avionics]," *Aerospace and Electronic Systems Magazine, IEEE,* vol. 18, pp. 30-36, 2003.

[Charchalakis'03] Periklis Charchalakis, George Valsamakis, R.M. Connor, et al., "MilCAN and Ethernet," presented at the 9th International CAN coference, Munich Germany, 2003.

Bibliography

[Rose'96] P. D. Rose, "Automotive and aerospace electronic systems. Dependability requirements," *Microelectronics and Reliability,* vol. 36, pp. 1923-1929, 1996.

[Bosch'91] Robert Bosch, "CAN Specification Version 2.0," ed, 1991.

[Kopetz'98] Hermann Kopetz, "A Comparison of CAN and TTP," in *In Proc. of the IFAC Distributed Computer Systems Workshop*, 1998, pp. 13--182.

[Çenesİz'04] Nilüfer Çenesİz and Murat Esin, "Controller area network (CAN) for computer integrated manufacturing systems," *Journal of Intelligent Manufacturing,* vol. 15, pp. 481-489, 2004.

[Koopman'02] P. Koopman, "Critical embedded automotive networks," *Micro, IEEE,* vol. 22, pp. 14-18, 2002.

[Gui Yun'01] Tian Gui Yun, "Design and implementation of distributed measurement systems using fieldbus-based intelligent sensors," *Instrumentation and Measurement, IEEE Transactions on,* vol. 50, pp. 1197-1202, 2001.

[Boys'04] Robert Boys, "Diagnostics and Prognostics for Military and Heavy Vehicles," Dearborn Group Inc2004.

[Tovar'01] Eduardo Tovar and Francisco Vasques, "Distributed computing for the factory-floor: a real-time approach using WorldFIP networks," *Computers in Industry,* vol. 44, pp. 11-31, 2001.

[Bucci'03] G. Bucci and C. Landi, "A distributed measurement architecture for industrial applications," *Instrumentation and Measurement, IEEE Transactions on,* vol. 52, pp. 165-174, 2003.

[Kunes'01] M. Kunes and T. Sauter, "Fieldbus-internet connectivity: the SNMP approach," *Industrial Electronics, IEEE Transactions on,* vol. 48, pp. 1248-1256, 2001.

[Stanton'96] Neville A. Stanton and Philip Marsden, "From fly-by-wire to drive-by-wire: Safety implications of automation in vehicles," *Safety Science,* vol. 24, pp. 35-49, 1996.

[Kyung Chang'03] Lee Kyung Chang, Kim Man Ho, Lee Suk, et al., "IEEE 1451 based smart module for in-vehicle networking systems of intelligent vehicles," in *Industrial Electronics Society, 2003. IECON '03. The 29th Annual Conference of the IEEE*, 2003, pp. 1796-1801 Vol.2.

[Ekiz'97] H. Ekiz, A. Kutlu. and E. T. Powner, "Implementation of CAN/CAN bridges in distributed environments and performance analysis of bridged CAN systems using SAE benchmark," in *Southeastcon '97. 'Engineering new New Century'., Proceedings. IEEE*, 1997, pp. 185-187.

[Kunert'97] O. Kunert and M. Zitterbart, "Interconnecting fieldbuses through ATM," in *Local Computer Networks, 1997. Proceedings., 22nd Annual Conference on*, 1997, pp. 538-544.

[Sveda'97] M. Sveda and F. Zezulka, "Interconnecting low-level fieldbusses," in *EUROMICRO 97. 'New Frontiers of Information Technology'., Proceedings of the 23rd EUROMICRO Conference*, 1997, pp. 614-620.

[Cavalieri'96] S. Cavalieri, A. Di Stefano. and O. Mirabella, "Mapping automotive process control on IEC/ISA FieldBus functionalities," *Computers in Industry,* vol. 28, pp. 233-250, 1996.

[Condor'00] Condor, "MIL-STD-1553 Protocol Tutorial," Condor Engineering2000.

[Koubias'95] S. A. Koubias and G. D. Papadopoulos, "Modern fieldbus communication architectures for real-time industrial applications," *Computers in Industry,* vol. 26, pp. 243-252, 1995.

[Lind'99] R. Lind, R. Schumacher, R. Reger, et al., "The Network Vehicle-a glimpse into the future of mobile multi-media," *Aerospace and Electronic Systems Magazine, IEEE,* vol. 14, pp. 27-32, 1999.

[Vitturi'01] S. Vitturi, "On the use of Ethernet at low level of factory communication systems," *Computer Standards & Interfaces,* vol. 23, pp. 267-277, 2001.

[Kibler'04] Thomas Kibler, Stefan Poferl, Gotthard Böck, et al., "Optical Data Buses for Automotive Applications," *J. Lightwave Technol.,* vol. 22, p. 2184, 2004.

[Fleming'01] W. J. Fleming, "Overview of automotive sensors," *Sensors Journal, IEEE,* vol. 1, pp. 296-308, 2001.

[Cavalier'96] S. Cavalier, A. Di Stefano, L. Lo Bello, et al., "Performance evaluation of FDDI-based fieldbus interconnection in factory automation environment," in *Industrial Electronics, 1996. ISIE '96., Proceedings of the IEEE International Symposium on*, 1996, pp. 1025-1030 vol.2.

[Papadopoulos'99] Y. Papadopoulos and J. A. McDermid, "The potential for a generic approach to certification of safety critical systems in the transportation sector," *Reliability Engineering & System Safety,* vol. 63, pp. 47-66, 1999.

[Giusto'04] Paolo Giusto and Thilo Demmeler, "Rapid design exploration of safety-critical distributed automotive applications via virtual integration platforms," *Journal of Systems and Software,* vol. 70, pp. 245-262, 2004.

[Flammini'02] A. Flammini, P. Ferrari, E. Sisinni, et al., "Sensor interfaces: from field-bus to Ethernet and Internet," *Sensors and Actuators A: Physical,* vol. 101, pp. 194-202, 2002.

[Simonds'03] C. Simonds, "Software for the next-generation automobile," *IT Professional,* vol. 5, pp. 7-11, 2003.

[Schumny'98b] Harald Schumny and Norbert Zisky, "A standard interface for site electronics based on modern fieldbus technology," *Computer Standards & Interfaces,* vol. 19, pp. 305-312, 1998b.

[Wood'00] G. Wood, "State of play [fieldbus technology]," *IEE Review,* vol. 46, pp. 26-28, 2000.

[Hammett'99] R. C. Hammett, "Ultra-reliable real-time control systems-future trends," *Aerospace and Electronic Systems Magazine, IEEE,* vol. 14, pp. 31-36, 1999.

[Heikkila'04] H. Heikkila, P. Eskelinen, P. Hautala, et al., "Upgrading armored vehicle sensor systems," *Aerospace and Electronic Systems Magazine, IEEE,* vol. 19, pp. 26-32, 2004.

[Stipidis'06] Elias Stipidis, Panagiotis Ioannis Oikonomidis. and George Valsamakis, "2004-06 Programmes Deliverable Technical Report, RT/COM/4/5008.04-06," ed, 2006.

[Ferreira'02] J. Ferreira, P. Pedreiras, L. Almeida, et al., "Achieving fault tolerance in FTT-CAN," in *Factory Communication Systems, 2002. 4th IEEE International Workshop on*, 2002, pp. 125-132.

[Haowei'07] Bai Haowei, "Analysis of a SAE AS5643 Mil-1394b Based High-Speed Avionics Network Architecture for Space and Defense Applications," in *Aerospace Conference, 2007 IEEE*, 2007, pp. 1-9.

[Kai'09] Wang Kai, Xu Aidong. and Wang Hong, "Avoiding the babbling idiot failure in a communication system based on flexible time division multiple access: A bus guardian solution," in *Industrial Electronics, 2009. ISIE 2009. IEEE International Symposium on*, 2009, pp. 1292-1297.

[Guo'07] Xiaosong Guo, Xingjie Pan, Chuanqiang Yu, et al., "Design of a CAN Bus Testing and Control System Based on Fault Tolerant Redundancy," in *Electronic Measurement and Instruments, 2007. ICEMI '07. 8th International Conference on*, 2007, pp. 1-888-1-892.

[Pickard'07] K. Pickard, T. Leopold, P. Muller, et al., "Electronic Failures and Monitoring Strategies in Automotive Control Units," in *Reliability and Maintainability Symposium, 2007. RAMS '07. Annual*, 2007, pp. 17-21.

[Kopetz'03a] H. Kopetz, "Fault containment and error detection in the time-triggered architecture," in *Autonomous Decentralized Systems, 2003. ISADS 2003. The Sixth International Symposium on*, 2003a, pp. 139-146.

[Jianmin'07] Duan Jianmin, Xiao Jinjun. and Zhang Mingjie, "Framework of CANopen Protocol for a Hybrid Electric Vehicle," in *Intelligent Vehicles Symposium, 2007 IEEE*, 2007, pp. 906-911.

[Hopkins'78] A. L. Hopkins, Jr., T. B. Smith, III. and J. H. Lala, "FTMP - A highly reliable fault-tolerant multiprocess for aircraft," *Proceedings of the IEEE,* vol. 66, pp. 1221-1239, 1978.

[Valsamakis'04] George Valsamakis, Periklis Charchalakis. and Elias Stipidis, "Management and configuration for MilCAN vetronic systems," presented at the 10th International CAN conference, 2004.

[Izosimov'06a] V. Izosimov, P. Pop, P. Eles, et al., "Mapping of Fault-Tolerant Applications with Transparency on Distributed Embedded Systems*," in *Digital System Design: Architectures, Methods and Tools, 2006. DSD 2006. 9th EUROMICRO Conference on*, 2006a, pp. 313-322.

[Buja'05] G. Buja, A. Zuccollo. and J. Pimentel, "Overcoming babbling-idiot failures in the FlexCAN architecture: a simple bus-guardian," in *Emerging Technologies and Factory Automation, 2005. ETFA 2005. 10th IEEE Conference on*, 2005, pp. 8 pp.-468.

[Webber'91] S. Webber and J. Beirne, "The Stratus architecture," in *Fault-Tolerant Computing, 1991. FTCS-21. Digest of Papers., Twenty-First International Symposium*, 1991, pp. 79-85.

[Izosimov'06b] V. Izosimov, P. Pop, P. Eles, et al., "Synthesis of fault-tolerant embedded systems with checkpointing and replication," in *Electronic Design, Test and Applications, 2006. DELTA 2006. Third IEEE International Workshop on*, 2006b, pp. 5 pp.-447.

[Bolchini'02] C. Bolchini, L. Pomante, F. Salice, et al., "A system level approach in designing dual-duplex fault tolerant embedded systems," in *On-Line Testing Workshop, 2002. Proceedings of the Eighth IEEE International*, 2002, pp. 32-36.

[Lee'01]B. H. Lee, "Using Bayes belief networks in industrial FMEA modeling and analysis," in *Reliability and Maintainability Symposium, 2001. Proceedings. Annual*, 2001, pp. 7-15.

[Charchalakis'06] Periklis Charchalakis, George Valsamakis, Elias Stipidis, et al., "VSI embedded network intergrated system for military vehicles," *Journal of Defence Science,* vol. 10, April 2005 2006.

[Philippi'03] Stephan Philippi, "Analysis of fault tolerance and reliability in distributed real-time system architectures," *Reliability Engineering & System Safety,* vol. 82, pp. 195-206, 2003.

[Kopetz'03b] H. Kopetz and G. Bauer, "The time-triggered architecture," *Proceedings of the IEEE,* vol. 91, pp. 112-126, 2003b.

[Karlsson'02] Annika Karlsson, "X-by-wire systems and time-triggered protocols," Uppsala University, 2002.

[Avizienis'04] A. Avizienis, J. C. Laprie, B. Randell, et al., "Basic concepts and taxonomy of dependable and secure computing," *Dependable and Secure Computing, IEEE Transactions on,* vol. 1, pp. 11-33, 2004.

[Barron'96] M. B. Barron and W. F. Powers, "The role of electronic controls for future automotive mechatronic systems," *Mechatronics, IEEE/ASME Transactions on,* vol. 1, pp. 80-88, 1996.

[Bauer'00] G. Bauer and H. Kopetz, "Transparent redundancy in the time-triggered architecture," in *Dependable Systems and Networks, 2000. DSN 2000. Proceedings International Conference on*, 2000, pp. 5-13.

[Boehm'88] B. W. Boehm, "A spiral model of software development and enhancement," *Computer,* vol. 21, pp. 61-72, 1988.

[Hess'05] R. A. Hess, "From Health and Usage Monitoring to Integrated Fleet Management - Evolving Directions for Rotorcraft," in *Aerospace Conference, 2005 IEEE*, 2005, pp. 1-6.

[Jakovljevic'06] M. Jakovljevic and M. Artner, "Protocol-Level System Health Monitoring and Redundancy Management for Integrated Vehicle Health Management," in *25th Digital Avionics Systems Conference, 2006 IEEE/AIAA*, 2006, pp. 1-7.

[Johannessen'01] Johannessen, "SIRIUS 2001 – A University Drive-by-Wire Project," Department of Computer Engineering, Chalmers University of Technology Goteborg, Sweden2001.

[Kopetz'94a] H. Kopetz, "The design of fault-tolerant real-time systems," in *EUROMICRO 94. System Architecture and Integration. Proceedings of the 20th EUROMICRO Conference.*, 1994a, pp. 4-9.

[Kopetz'04] H. Kopetz, "An integrated architecture for dependable embedded systems," in *Reliable Distributed Systems, 2004. Proceedings of the 23rd IEEE International Symposium on*, 2004, pp. 160-161.

[Kopetz'94b] H. Kopetz and G. Grunsteidl, "TTP-a protocol for fault-tolerant real-time systems," *Computer,* vol. 27, pp. 14-23, 1994b.

[Kopetz'99] H. Kopetz and D. Millinger, "The transparent implementation of fault tolerance in the time-triggered architecture," in *Dependable Computing for Critical Applications 7, 1999*, 1999, pp. 191-205.

[Lincoln'95] P. Lincoln and J. Rushby, "A Formally Verified Algorithm for Interactive Consistency Under a Hybrid Fault Model," in *Fault-Tolerant Computing, 1995, ' Highlights from Twenty-Five Years'., Twenty-Fifth International Symposium on*, 1995, p. 438.

[MoD'06] UK Ministry of Defence MoD, "Defence Technology Strategy 2006," 2006.

[Murray'02] C Murray, *A Guide to Successful Software Development*. NY: Addison-Wesley, 2002.

[Rushby'96] J. Rushby, "Reconfiguration and transient recovery in state machine architectures," in *Fault Tolerant Computing, 1996., Proceedings of Annual Symposium on*, 1996, pp. 6-15.

[Rushby'99] J. Rushby, "Systematic formal verification for fault-tolerant time-triggered algorithms," *Software Engineering, IEEE Transactions on,* vol. 25, pp. 651-660, 1999.

[White'08] Tony White, "The By-Wire Experience," presented at the TTTech Symposium, Ultra Electronics, 2008.

[Wideman'03] M Wideman, "Software Development and Linearity," *ICFAI PRESS Hyderabad,* 2003.

Appendix

MilCAN Reconfiguration Protocol

Function	Primary ID	Secondary ID	Direction	DLC
Start firmware load	0xBE	0x1E	M > S	3
	Payload	**Data**		**Description**
	0	<Slave ID>		
	1	<Start Sequence Number>		
	2	<Firmware Version>		
	3			
	4			
	5			
	6			
	7			

Function	Primary ID	Secondary ID	Direction	DLC
Start firmware load (ACK)	0xBE	0x1F	S > M	3
	Payload	**Data**		**Description**
	0	<Slave ID>		
	1	<Start Sequence Number>		
	2	<Firmware Version>		
	3			
	4			
	5			
	6			
	7			

Function	Primary ID	Secondary ID	Direction	DLC
Address set	0xBE	0x20	M > S	6
	Payload	**Data**		**Description**
	0	<Slave ID>		
	1	<Sequence Number>		
	2	<Data Size>		
	3	<Address Upper Bytes>		
	4	<Address Lower Bytes>		
	5	<Checksum>		
	6			
	7			

Function	Primary ID	Secondary ID	Direction	DLC
Address set (ACK)	0xBE	0x21	S > M	6
	Payload	**Data**		**Description**
	0	<Slave ID>		
	1	<Sequence Number>		
	2	<Data Size>		
	3	<Address Upper Bytes>		
	4	<Address Lower Bytes>		
	5	<Checksum>		
	6			
	7			

Function	Primary ID	Secondary ID	Direction	DLC
Programming data	0xBE	0x22	M > S	8
	Payload	**Data**		**Description**
	0	<Slave ID>		
	1	<Sequence Number>		
	2	<Data Upper Bytes>		
	3			
	4			
	5			
	6			
	7	<Data Lower Bytes>		

Function	Primary ID	Secondary ID	Direction	DLC
Programming data (ACK)	0xBE	0x23	S > M	8
	Payload	**Data**		**Description**
	0	<Slave ID>		
	1	<Sequence Number>		
	2	<Data Upper Bytes>		
	3			
	4			
	5			
	6			
	7	<Data Lower Bytes>		

Function	Primary ID	Secondary ID	Direction	DLC
Enter programming mode	0xBE	0x24	M > S	1
	Payload	**Data**		**Description**
	0	<Slave ID>		
	1			
	2			
	3			
	4			
	5			
	6			
	7			

Function	Primary ID	Secondary ID	Direction	DLC
Enter programming mode (ACK)	0xBE	0x25	S > M	1
	Payload	**Data**		**Description**
	0	<Slave ID>		
	1			
	2			
	3			
	4			
	5			
	6			
	7			

Function	Primary ID	Secondary ID	Direction	DLC
Erasing app sectors	0xBE	0x26	M > S	1
	Payload	**Data**		**Description**
	0	<Slave ID>		
	1			
	2			
	3			
	4			
	5			
	6			
	7			

Function	Primary ID	Secondary ID	Direction	DLC
Erasing app sectors (ACK)	0xBE	0x27	S > M	1
	Payload	**Data**		**Description**
	0	<Slave ID>		
	1			
	2			
	3			
	4			
	5			
	6			
	7			

Function	Primary ID	Secondary ID	Direction	DLC
Erasing chip	0xBE	0x28	M > S	1
	Payload	**Data**		**Description**
	0	<Slave ID>		
	1			
	2			
	3			
	4			
	5			
	6			
	7			

Function	Primary ID	Secondary ID	Direction	DLC
Erasing chip (ACK)	0xBE	0x29	S > M	1
	Payload	**Data**		**Description**
	0	<Slave ID>		
	1			
	2			
	3			
	4			
	5			
	6			
	7			

Function	Primary ID	Secondary ID	Direction	DLC
Software reset	0xBE	0x2A	M > S	1
	Payload	**Data**		**Description**
	0	<Slave ID>		
	1			
	2			
	3			
	4			
	5			
	6			
	7			

Function	Primary ID	Secondary ID	Direction	DLC
Software reset (ACK)	0xBE	0x2B	S > M	1
	Payload	**Data**		**Description**
	0	<Slave ID>		
	1			
	2			
	3			
	4			
	5			
	6			
	7			

Function	Primary ID	Secondary ID	Direction	DLC
Checksum	0xBE	0x2C	M > S	3
	Payload	**Data**		**Description**
	0	<Slave ID>		
	1	<Checksum upper>		
	2	<Checksum lower>		
	3			
	4			
	5			
	6			
	7			

Function	Primary ID	Secondary ID	Direction	DLC
Checksum (ACK)	0xBE	0x2D	S > M	3
	Payload	**Data**		**Description**
	0	<Slave ID>		
	1	<Checksum upper>		
	2	<Checksum lower>		
	3			
	4			
	5			
	6			
	7			

Function	Primary ID	Secondary ID	Direction	DLC
Status checking	0xBE	0x2E	M > S	1
	Payload	**Data**		**Description**
	0	<Slave ID>		
	1			
	2			
	3			
	4			
	5			
	6			
	7			

Function	Primary ID	Secondary ID	Direction	DLC
Status checking (result)	0xBE	0x2F	S > M	3
	Payload	**Data**		**Description**
	0	<Slave ID>		
	1	<MilCAN mode>		
	2	<Node mode>		
	3			
	4			
	5			
	6			
	7			

Function	Primary ID	Secondary ID	Direction	DLC
Version checking	0xBE	0x30	M > S	1
	Payload	**Data**		**Description**
	0	<Slave ID>		
	1			
	2			
	3			
	4			
	5			
	6			
	7			

Function	Primary ID	Secondary ID	Direction	DLC
Version checking (result)	0xBE	0x31	S > M	8
	Payload	**Data**		**Description**
	0	<Slave ID>		
	1	<Version>		
	2	<Version>		
	3	<Version>		
	4	<Version>		
	5	<Version>		
	6	<Version>		
	7	<Version>		

Function	Primary ID	Secondary ID	Direction	DLC
Node ID change	0xBE	0x32	M > S	7
	Payload	**Data**		**Description**
	0	<Slave ID>		
	1	<MILCAN_ID1>		
	2	<MILCAN_ID2>		
	3	<NODE_SERIAL>		
	4	<NODE_SERIAL>		
	5	<NODE_SERIAL>		
	6	<NODE_SERIAL>		
	7			

Function	Primary ID	Secondary ID	Direction	DLC
Node ID change (result)	0xBE	0x33	S > M	7
	Payload	**Data**		**Description**
	0	<Slave ID>		
	1	<MILCAN_ID1>		
	2	<MILCAN_ID2>		
	3	<NODE_SERIAL>		
	4	<NODE_SERIAL>		
	5	<NODE_SERIAL>		
	6	<NODE_SERIAL>		
	7			

Function	Primary ID	Secondary ID	Direction	DLC
Node speed change	0xBE	0x34	M > S	5
	Payload	**Data**		**Description**
	0	<Slave ID>		
	1	<NODE_SPEED_A_1>		for 1000 kbits
	2	<NODE_SPEED_A_2>		
	3	<NODE_SPEED_B_1>		
	4	<NODE_SPEED_B_1>		
	5			
	6			
	7			

Function	Primary ID	Secondary ID	Direction	DLC
Node speed change (result)	0xBE	0x35	S > M	5
	Payload	**Data**		**Description**
	0	<Slave ID>		
	1	<NODE_SPEED_A_1>		
	2	<NODE_SPEED_A_2>		
	3	<NODE_SPEED_B_1>		
	4	<NODE_SPEED_B_1>		
	5			
	6			
	7			

Function	Primary ID	Secondary ID	Direction	DLC
Message configuration	0xBE	0x50	M > S	8
	Payload	**Data**		**Description**
	0	<Slave ID>		
	1	<Primary ID>		
	2	<Secondary ID>		
	3	<New Priority>		
	4	<New Primary ID>		
	5	<New Secondary ID>		
	6	<New Start Frame>		
	7	<New Cycle>		

Function	Primary ID	Secondary ID	Direction	DLC
Message configuration (ACK)	0xBE	0x51	S > M	8
	Payload	**Data**		**Description**
	0	<Slave ID>		
	1	<Primary ID>		
	2	<Secondary ID>		
	3	<New Priority>		
	4	<New Primary ID>		
	5	<New Secondary ID>		
	6	<New Start Frame>		
	7	<New Cycle>		

Function	Primary ID	Secondary ID	Direction	DLC
Message configuration save	0xBE	0x52	M > S	1
	Payload	**Data**		**Description**
	0	<Slave ID>		
	1			
	2			
	3			
	4			
	5			
	6			
	7			

Function	Primary ID	Secondary ID	Direction	DLC
Message configuration save (ACK)	0xBE	0x53	S > M	1
	Payload	**Data**		**Description**
	0	<Slave ID>		
	1			
	2			
	3			
	4			
	5			
	6			
	7			

MilCAN Reconfiguration test result

Automotive segment message latencies; MilCAN segment 1000kbit
MilCAN backbone 1000kbit, routing MilCAN backbone

		SEGA to BBONE		BBONE to SEGU		SEGA to SEGU	
ID	Message	high HRT	low HRT	high HRT	low HRT	high HRT	low HRT
0x3e30	MAPED1	0.0343	0.0395	0.0209	0.0255	0.0556	0.0636
0x3e32	MAPED2	0.0357	0.0403	0.0209	0.0259	0.0569	0.0648
0x5e30	MAPED3	0.0455	0.0392	0.0295	0.0273	0.0723	0.0652
0x3e34	MASTR1	0.0342	0.0403	0.021	0.025	0.0553	0.0647
0x3e36	MASTR2	0.0408	0.0427	0.0235	0.0265	0.0652	0.0686
0x5830	MASTR3	0.04	0.0434	0.0233	0.0282	0.0644	0.0712
0x5832	MASTR4	0.0417	0.0435	0.0238	0.0282	0.067	0.0712
0x5834	MASTR5	0.0441	0.0436	0.028	0.0283	0.0696	0.0713
0x4430	MASTR6	0.0404	0.0393	0.0238	0.0278	0.0655	0.0661
0xd220	MAWHE1	0.0347	0.0381	0.0212	0.0255	0.0562	0.0651
0xd221	MAWHE2	0.0269	0.0388	0.0211	0.0244	0.035	0.0649
0xf020	MAENG1	0.0348	0.0404	0.0209	0.0254	0.0557	0.0652
0xf021	MAENG2	0.0356	0.0387	0.021	0.0257	0.0566	0.0659
0xf022	MAENG3	0.0422	0.0395	0.0238	0.0267	0.0665	0.0682

Utilities segment message latencies; MilCAN segment 1000kbit
MilCAN backbone 1000kbit, routing MilCAN backbone

		SEGU to BBONE		BBONE to SEGA		SEGU to SEGA	
ID	Message	high HRT	low HRT	high HRT	low HRT	high HRT	low HRT
0x5501	MUGPS1	0.0181	0.0247	0.0368	0.0457	0.0544	0.071
0x5502	MUGPS2	0.0209	0.0245	0.0376	0.046	0.0567	0.0709
0x5503	MUGPS3	0.0219	0.0245	0.0383	0.0461	0.0587	0.0709
0x5030	MUPWR1	0.0232	0.0256	0.0395	0.0442	0.0596	0.0659
0x5936	MUSNS1	0.0249	0.0197	0.043	0.0429	0.0685	0.0603

Automotive segment message latencies; MilCAN segment 1000kbit
MilCAN backbone 1000kbit, routing Ethernet backbone

		SEGA to SEGU	
ID	Message	high HRT	low HRT
0x3e30	MAPED1	0.0368	0.0323
0x3e32	MAPED2	0.0384	0.0359
0x5e30	MAPED3	0.0469	0.039
0x3e34	MASTR1	0.0365	0.0312
0x3e36	MASTR2	0.0465	0.0408
0x5830	MASTR3	0.0461	0.0383
0x5832	MASTR4	0.0471	0.0409
0x5834	MASTR5	0.048	0.0427
0x4430	MASTR6	0.0462	0.0412
0xd220	MAWHE1	0.0377	0.0337
0xd221	MAWHE2	0.0354	0.0321
0xf020	MAENG1	0.037	0.0325
0xf021	MAENG2	0.0385	0.0342
0xf022	MAENG3	0.0482	0.0398

Utilities segment message latencies; MilCAN segment 1000kbit
MilCAN backbone 1000kbit, routing Ethernet backbone

		SEGU to SEGA	
ID	Message	high HRT	low HRT
0x5501	MUGPS1	0.0296	0.0252
0x5502	MUGPS2	0.0315	0.0387
0x5503	MUGPS3	0.0314	0.0402
0x5030	MUPWR1	0.0301	0.0403
0x5936	MUSNS1	0.0186	0.0252

Automotive segment message latencies; MilCAN segment 1000kbit
MilCAN backbone 1000kbit, routing MilCAN/Ethernet backbone

		SEGA to SEGU	
ID	Message	high HRT	low HRT
0x3e30	MAPED1	0.0519	0.0592
0x3e32	MAPED2	0.0535	0.0597
0x5e30	MAPED3	0.0396	0.0298
0x3e34	MASTR1	0.0518	0.0623
0x3e36	MASTR2	0.0644	0.0646
0x5830	MASTR3	0.0344	0.0353
0x5832	MASTR4	0.0366	0.0358
0x5834	MASTR5	0.038	0.0357
0x4430	MASTR6	0.0647	0.0576
0xd220	MAWHE1	0.0534	0.0652
0xd221	MAWHE2	0.0313	0.0631
0xf020	MAENG1	0.0522	0.0627
0xf021	MAENG2	0.0533	0.0638
0xf022	MAENG3	0.0663	0.0645

Utilities segment message latencies; MilCAN segment 1000kbit
MilCAN backbone 1000kbit, routing MilCAN/Ethernet backbone

		SEGU to SEGA	
ID	Message	high HRT	low HRT
0x5501	MUGPS1	0.0672	0.0724
0x5502	MUGPS2	0.0681	0.0724
0x5503	MUGPS3	0.0685	0.0723
0x5030	MUPWR1	0.0329	0.0391
0x5936	MUSNS1	0.0305	0.0346

Automotive segment message latencies; MilCAN segment 1000kbit
MilCAN backbone 500kbit, routing MilCAN backbone

		SEGA to BBONE		BBONE to SEGU		SEGA to SEGU	
ID	Message	high HRT	low HRT	high HRT	low HRT	high HRT	low HRT
0x3e30	MAPED1	0.0343	0.0394	0.0246	0.0254	0.0547	0.0639
0x3e32	MAPED2	0.0356	0.0401	0.0206	0.0258	0.0562	0.0652
0x5e30	MAPED3	0.044	0.0395	0.0279	0.027	0.0704	0.065
0x3e34	MASTR1	0.0339	0.0412	0.0208	0.0256	0.0546	0.0663
0x3e36	MASTR2	0.0404	0.0432	0.023	0.0268	0.0631	0.0694
0x5830	MASTR3	0.0402	0.0431	0.0229	0.0279	0.0616	0.0701
0x5832	MASTR4	0.0413	0.0434	0.0241	0.0281	0.0641	0.0702
0x5834	MASTR5	0.0423	0.0437	0.0257	0.0281	0.0673	0.0703
0x4430	MASTR6	0.0401	0.0399	0.0229	0.0285	0.0634	0.0659
0xd220	MAWHE1	0.0345	0.0417	0.0205	0.0277	0.0543	0.0671
0xd221	MAWHE2	0.0151	0.0426	0.0207	0.0253	0.0329	0.0642
0xf020	MAENG1	0.0345	0.0416	0.0208	0.0255	0.0552	0.0662
0xf021	MAENG2	0.0356	0.0411	0.0208	0.0262	0.056	0.0677
0xf022	MAENG3	0.0419	0.0418	0.0231	0.0266	0.0646	0.0697

Utilities segment message latencies; MilCAN segment 1000kbit
MilCAN backbone 500kbit, routing MilCAN backbone

		SEGU to BBONE		BBONE to SEGA		SEGU to SEGA	
ID	Message	high HRT	low HRT	high HRT	low HRT	high HRT	low HRT
0x5501	MUGPS1	0.0182	0.0238	0.0367	0.0485	0.055	0.0651
0x5502	MUGPS2	0.0214	0.0238	0.0377	0.0469	0.0593	0.0653
0x5503	MUGPS3	0.0233	0.0242	0.038	0.0469	0.06	0.0652
0x5030	MUPWR1	0.0237	0.0215	0.0379	0.046	0.0593	0.0595
0x5936	MUSNS1	0.0246	0.0201	0.0419	0.042	0.066	0.0597

Automotive segment message latencies; MilCAN segment 1000kbit
MilCAN backbone 500kbit, routing MilCAN/Ethernet backbone

		SEGA to SEGU	
ID	Message	high HRT	low HRT
0x3e30	MAPED1	0.0516	0.0593
0x3e32	MAPED2	0.0534	0.0598
0x5e30	MAPED3	0.0382	0.0294
0x3e34	MASTR1	0.0512	0.0608
0x3e36	MASTR2	0.0631	0.0627
0x5830	MASTR3	0.0352	0.0353
0x5832	MASTR4	0.036	0.0354
0x5834	MASTR5	0.0366	0.0353
0x4430	MASTR6	0.0638	0.0623
0xd220	MAWHE1	0.0524	0.0627
0xd221	MAWHE2	0.0306	0.0614
0xf020	MAENG1	0.0521	0.061
0xf021	MAENG2	0.0534	0.0612
0xf022	MAENG3	0.0648	0.0617

Utilities segment message latencies; MilCAN segment 1000kbit MilCAN backbone 500kbit, routing MilCAN/Ethernet backbone

		SEGU to SEGA	
ID	Message	high HRT	low HRT
0x5501	MUGPS1	0.0656	0.0671
0x5502	MUGPS2	0.0662	0.0686
0x5503	MUGPS3	0.0665	0.0687
0x5030	MUPWR1	0.0283	0.0294
0x5936	MUSNS1	0.0288	0.0297

Automotive segment message latencies; MilCAN segment 1000kbit MilCAN backbone 250kbit, routing MilCAN backbone

		SEGA to BBONE		BBONE to SEGU		SEGA to SEGU	
ID	Message	high HRT	low HRT	high HRT	low HRT	high HRT	low HRT
0x3e30	MAPED1	0.0351	0.0398	0.0207	0.0256	0.0552	0.0641
0x3e32	MAPED2	0.0365	0.0408	0.0209	0.0258	0.0567	0.0656
0x5e30	MAPED3	0.0432	0.0399	0.027	0.0277	0.0693	0.0653
0x3e34	MASTR1	0.0348	0.0412	0.0207	0.0256	0.0549	0.066
0x3e36	MASTR2	0.0403	0.0428	0.0232	0.0271	0.0626	0.0688
0x5830	MASTR3	0.0396	0.0432	0.0248	0.0288	0.0608	0.0715
0x5832	MASTR4	0.0406	0.0436	0.0257	0.0285	0.064	0.0719
0x5834	MASTR5	0.0421	0.044	0.0275	0.0283	0.0687	0.0719
0x4430	MASTR6	0.0405	0.0384	0.0238	0.0295	0.0627	0.0672
0xd220	MAWHE1	0.0354	0.041	0.0212	0.0283	0.0555	0.0661
0xd221	MAWHE2	0.0298	0.0416	0.0208	0.0251	0.0346	0.0663
0xf020	MAENG1	0.0352	0.0415	0.0209	0.0256	0.0553	0.0663
0xf021	MAENG2	0.0363	0.041	0.0211	0.0253	0.0564	0.0678
0xf022	MAENG3	0.0416	0.0413	0.0227	0.0261	0.0638	0.0687

Utilities segment message latencies; MilCAN segment 1000kbit MilCAN backbone 250kbit, routing MilCAN backbone

		SEGU to BBONE		BBONE to SEGA		SEGU to SEGA	
ID	Message	high HRT	low HRT	high HRT	low HRT	high HRT	low HRT
0x5501	MUGPS1	0.0193	0.0262	0.0355	0.0386	0.0515	0.0686
0x5502	MUGPS2	0.0221	0.0266	0.0363	0.0386	0.056	0.0687
0x5503	MUGPS3	0.0233	0.0269	0.0366	0.0382	0.0586	0.069
0x5030	MUPWR1	0.0237	0.0275	0.0346	0.038	0.0607	0.0619
0x5936	MUSNS1	0.0251	0.0207	0.0412	0.0411	0.0665	0.0593

Automotive segment message latencies; MilCAN segment 1000kbit
MilCAN backbone 250kbit, routing MilCAN/Ethernet backbone

		SEGA to SEGU	
ID	Message	high HRT	low HRT
0x3e30	MAPED1	0.0532	0.0603
0x3e32	MAPED2	0.0548	0.0609
0x5e30	MAPED3	0.0377	0.0298
0x3e34	MASTR1	0.0529	0.0615
0x3e36	MASTR2	0.0608	0.0636
0x5830	MASTR3	0.0335	0.0352
0x5832	MASTR4	0.0344	0.0353
0x5834	MASTR5	0.0366	0.0352
0x4430	MASTR6	0.0612	0.065
0xd220	MAWHE1	0.0543	0.0665
0xd221	MAWHE2	0.0315	0.063
0xf020	MAENG1	0.0535	0.0623
0xf021	MAENG2	0.0549	0.0628
0xf022	MAENG3	0.0628	0.0638

Utilities segment message latencies; MilCAN segment 1000kbit
MilCAN backbone 250kbit, routing MilCAN/Ethernet backbone

		SEGU to SEGA	
ID	Message	high HRT	low HRT
0x5501	MUGPS1	0.0645	0.0709
0x5502	MUGPS2	0.0656	0.0736
0x5503	MUGPS3	0.0667	0.0749
0x5030	MUPWR1	0.0297	0.0408
0x5936	MUSNS1	0.029	0.0359

Automotive segment message latencies; MilCAN segment 500kbit
MilCAN backbone 500kbit, routing MilCAN backbone

		SEGA to BBONE		BBONE to SEGU		SEGA to SEGU	
ID	Message	high HRT	low HRT	high HRT	low HRT	high HRT	low HRT
0x3e30	MAPED1	0.0399	0.0253	0.0215	0.0183	0.0626	0.0431
0x3e32	MAPED2	0.0409	0.0263	0.0219	0.0196	0.0639	0.0453
0x5e30	MAPED3	0.0502	0.021	0.0208	0.0182	0.0738	0.0389
0x3e34	MASTR1	0.0435	0.0309	0.0212	0.023	0.0654	0.0536
0x3e36	MASTR2	0.0474	0.0323	0.0215	0.0254	0.0699	0.0565
0x5830	MASTR3	0.0437	0.0293	0.02	0.0211	0.069	0.0501
0x5832	MASTR4	0.0456	0.0295	0.02	0.0212	0.0691	0.0502
0x5834	MASTR5	0.0457	0.0297	0.0206	0.0214	0.0693	0.0505
0x4430	MASTR6	0.0458	0.0327	0.0219	0.0231	0.0681	0.0555
0xd220	MAWHE1	0.0452	0.0363	0.0209	0.0241	0.066	0.06
0xd221	MAWHE2	0.0433	0.0305	0.0216	0.0245	0.0576	0.0551
0xf020	MAENG1	0.0434	0.0314	0.0214	0.0237	0.0661	0.0539
0xf021	MAENG2	0.0444	0.0347	0.0222	0.0246	0.0684	0.0584
0xf022	MAENG3	0.0484	0.0346	0.0224	0.0251	0.0721	0.0589

Utilities segment message latencies; MilCAN segment 500kbit
MilCAN backbone 500kbit, routing MilCAN backbone

		SEGU to BBONE		BBONE to SEGA		SEGU to SEGA	
ID	Message	high HRT	low HRT	high HRT	low HRT	high HRT	low HRT
0x5501	MUGPS1	0.0187	0.0235	0.0497	0.0292	0.0667	0.0533
0x5502	MUGPS2	0.0205	0.0236	0.0504	0.0291	0.0677	0.0532
0x5503	MUGPS3	0.0207	0.0209	0.0494	0.029	0.0666	0.0504
0x5030	MUPWR1	0.0182	0.0254	0.0552	0.027	0.0699	0.0487
0x5936	MUSNS1	0.0185	0.0158	0.0471	0.0217	0.0642	0.0358

Automotive segment message latencies; MilCAN segment 500kbit
MilCAN backbone 500kbit, routing Ethernet backbone

		SEGA to SEGU	
ID	Message	high HRT	low HRT
0x3e30	MAPED1	0.0408	0.0318
0x3e32	MAPED2	0.0421	0.0347
0x5e30	MAPED3	0.0562	0.03
0x3e34	MASTR1	0.0462	0.0401
0x3e36	MASTR2	0.0534	0.0396
0x5830	MASTR3	0.0566	0.0423
0x5832	MASTR4	0.0578	0.0419
0x5834	MASTR5	0.0578	0.0356
0x4430	MASTR6	0.0523	0.0412
0xd220	MAWHE1	0.0442	0.043
0xd221	MAWHE2	0.0474	0.0421
0xf020	MAENG1	0.0465	0.0325
0xf021	MAENG2	0.049	0.034
0xf022	MAENG3	0.0546	0.0398

Utilities segment message latencies; MilCAN segment 500kbit
MilCAN backbone 500kbit, routing Ethernet backbone

		SEGU to SEGA	
ID	Message	high HRT	low HRT
0x5501	MUGPS1	0.0212	0.0197
0x5502	MUGPS2	0.0211	0.02
0x5503	MUGPS3	0.0224	0.0155
0x5030	MUPWR1	0.0229	0.0176
0x5936	MUSNS1	0.0167	0.0125

Automotive segment message latencies; MilCAN segment 500kbit
MilCAN backbone 500kbit, routing MilCAN/Ethernet backbone

		SEGA to SEGU	
ID	Message	high HRT	low HRT
0x3e30	MAPED1	0.0607	0.0381
0x3e32	MAPED2	0.0612	0.0408
0x5e30	MAPED3	0.0408	0.0158
0x3e34	MASTR1	0.0656	0.0469
0x3e36	MASTR2	0.0695	0.0513
0x5830	MASTR3	0.042	0.0247
0x5832	MASTR4	0.0421	0.0256
0x5834	MASTR5	0.0422	0.0262
0x4430	MASTR6	0.0683	0.0476
0xd220	MAWHE1	0.0647	0.0555
0xd221	MAWHE2	0.0629	0.0481
0xf020	MAENG1	0.0662	0.048
0xf021	MAENG2	0.0691	0.0552
0xf022	MAENG3	0.0722	0.0555

Utilities segment message latencies; MilCAN segment 500kbit
MilCAN backbone 500kbit, routing MilCAN/Ethernet backbone

		SEGU to SEGA	
ID	Message	high HRT	low HRT
0x5501	MUGPS1	0.0659	0.057
0x5502	MUGPS2	0.0662	0.0569
0x5503	MUGPS3	0.0657	0.0543
0x5030	MUPWR1	0.0305	0.0568
0x5936	MUSNS1	0.025	0.0268

Automotive segment message latencies; MilCAN segment 500kbit
MilCAN backbone 250kbit, routing MilCAN backbone

		SEGA to BBONE		BBONE to SEGU		SEGA to SEGU	
ID	Message	high HRT	low HRT	high HRT	low HRT	high HRT	low HRT
0x3e30	MAPED1	0.0418	0.0254	0.0231	0.0175	0.0635	0.0446
0x3e32	MAPED2	0.0434	0.0267	0.0233	0.0182	0.0649	0.0462
0x5e30	MAPED3	0.0509	0.0207	0.0236	0.0175	0.0707	0.039
0x3e34	MASTR1	0.0447	0.0305	0.0228	0.0212	0.0664	0.0543
0x3e36	MASTR2	0.0487	0.0324	0.0238	0.023	0.07	0.0575
0x5830	MASTR3	0.0446	0.0293	0.0228	0.0202	0.0638	0.0513
0x5832	MASTR4	0.0453	0.0298	0.0225	0.0205	0.0674	0.0521
0x5834	MASTR5	0.0456	0.03	0.0227	0.0207	0.0692	0.0524
0x4430	MASTR6	0.0472	0.0318	0.024	0.0221	0.0696	0.0554
0xd220	MAWHE1	0.0465	0.0351	0.0228	0.0236	0.0658	0.0588
0xd221	MAWHE2	0.0448	0.0315	0.0233	0.0224	0.0642	0.0574
0xf020	MAENG1	0.0457	0.0313	0.0229	0.0222	0.0669	0.0555
0xf021	MAENG2	0.0468	0.0352	0.0232	0.024	0.0692	0.061
0xf022	MAENG3	0.0502	0.0346	0.0237	0.0247	0.0715	0.0611

Utilities segment message latencies; MilCAN segment 500kbit
MilCAN backbone 250kbit, routing MilCAN backbone

		SEGU to BBONE		BBONE to SEGA		SEGU to SEGA	
ID	Message	high HRT	low HRT	high HRT	low HRT	high HRT	low HRT
0x5501	MUGPS1	0.0189	0.0205	0.0455	0.0386	0.063	0.062
0x5502	MUGPS2	0.0217	0.0209	0.0471	0.0385	0.0694	0.0621
0x5503	MUGPS3	0.0239	0.0199	0.045	0.0382	0.0699	0.0593
0x5030	MUPWR1	0.0229	0.0128	0.0426	0.0348	0.0691	0.0535
0x5936	MUSNS1	0.0195	0.0153	0.0464	0.021	0.0644	0.037

Automotive segment message latencies; MilCAN segment 500kbit
MilCAN backbone 250kbit, routing MilCAN/Ethernet backbone

		SEGA to SEGU	
ID	Message	high HRT	low HRT
0x3e30	MAPED1	0.0621	0.0385
0x3e32	MAPED2	0.0634	0.0401
0x5e30	MAPED3	0.04	0.0162
0x3e34	MASTR1	0.0668	0.0468
0x3e36	MASTR2	0.07	0.0504
0x5830	MASTR3	0.041	0.0238
0x5832	MASTR4	0.0411	0.0243
0x5834	MASTR5	0.0412	0.0245
0x4430	MASTR6	0.0709	0.046
0xd220	MAWHE1	0.0675	0.0525
0xd221	MAWHE2	0.0507	0.049
0xf020	MAENG1	0.0673	0.0484
0xf021	MAENG2	0.0714	0.054
0xf022	MAENG3	0.0735	0.0539

Utilities segment message latencies; MilCAN segment 500kbit
MilCAN backbone 250kbit, routing MilCAN/Ethernet backbone

		SEGU to SEGA	
ID	Message	high HRT	low HRT
0x5501	MUGPS1	0.0644	0.0472
0x5502	MUGPS2	0.0706	0.0472
0x5503	MUGPS3	0.071	0.0443
0x5030	MUPWR1	0.0269	0.0319
0x5936	MUSNS1	0.0249	0.0258