FAKULTÄT FÜR !NFORMATIK
Faculty of Informatics

Experimental Framework for Controller Area Network based on a Multi-Processor-System-on-a-Chip

DIPLOMARBEIT

zur Erlangung des akademischen Grades

Diplom-Ingenieur

im Rahmen des Studiums

Technische Informatik

eingereicht von

Walther Operenyi
Matrikelnummer 0407269

an der
Fakultät für Informatik der Technischen Universität Wien

Betreuung
Betreuer/in: Prof. Dipl.-Ing. Dr.techn. Roman Obermaisser
Mitwirkung: Univ.Ass. Dipl.-Ing. Roland Kammerer

Wien, 03.12.2012

(Unterschrift Verfasser) (Unterschrift Betreuer/in)

„Hiermit erkläre ich, dass ich diese Arbeit selbständig verfasst habe, dass ich die verwendeten Quellen und Hilfsmittel vollständig angegeben habe und dass ich die Stellen der Arbeit – einschließlich Tabellen, Karten und Abbildungen –, die anderen Werken oder dem Internet im Wortlaut oder dem Sinn nach entnommen sind, auf jeden Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht habe."

Wien am 03.12.2012

Abstract

Controller Area Network (CAN) is used for sending and receiving short real-time messages up to 1 Mbit/s. CAN is mainly used in the car industry to interconnect Electronic Control Units (ECUs) so that the ECUs communicate via messages. Each message has a unique id and a payload. Furthermore, each message with a unique id is sent repeatedly on demand to transport control information via its payload. The control information is generated and accepted by control tasks. Each task fulfills a well-defined operation purpose and is executed by an ECU. Therefore the unique id of a message definitely tags the control information in order to address tasks and ECUs. In addition, the message id determines the priority of a message. Each CAN message that is ready for transmission starts to emit if no message is in transmission. As multiple messages start to emit simultaneously the priority of a message determines its transport so that the message with the highest priority transmits and the remaining messages stop to emit. The remaining messages are sent afterwards. Thus the transmission times (difference between message generation and reception) of each message with a unique id vary.

Typically, many tasks in a CAN system have to concurrently exchange data and control information in real-time. Low transmission latencies and a low variability of the transmission latencies are important to ensure high control performance. In this diploma thesis an experimental evaluation technique is presented to determine statistical data of the transmission behaviour in CAN such as the maximum, minimum and average transmission latencies. Therefore a hardware platform was developed that consists of processors that emulate ECUs. Furthermore, tasks are simulated on each processor which generates messages of the same unique id randomly but within limited time intervals. These messages are transmitted via CAN and the transmission times are measured to derive statistical data. In addition, auxiliary parameters are collected such as number of sent and received messages at each particular processor. Furthermore, a complete toolchain is presented to design test configurations and analyze statistical data.

CAN has a good average transmission time behaviour but the transmission times are sometimes extremely delayed. In general, the transmission times are reduced if the collision probability is decreased. The collision probability is determined by the number of messages and by the send frequency of each message with a unique id. Thus the collision probability determines the utilization of the transmission medium.

Kurzfassung

Controller Area Network (CAN) dient zum Senden und Empfangen von kurzen Echtzeitnachrichten bei einem Datendurchsatz von bis zu 1 Mbit/s. Die Automobilindustrie setzt CAN für die Vernetzung von Electronic Control Units (ECUs) ein. Die Vernetzung ist mittels Nachrichtenübertragung realisiert. Eine Nachricht besteht aus einer eindeutigen Kennung und Nutzdaten. Eine Nachricht wird bei Bedarf instanziert und übertragen um Kontrollinformation zu senden. Kontrollinformation wird von Prozessen generiert und entgegengenommen und dient zum Regeln und Steuern. Ein Prozess erfüllt einen eindeutig definierten Zweck und wird auf einer ECU ausgeführt. Hierbei bezeichnet die Kennung die Nutzdaten um Prozesse und ECUs zu adressieren. Darüber hinaus bestimmt die Kennung die Priorität einer Nachricht. Eine Nachrichtenübertragung beginnt falls keine Nachtrichtenübertragung stattfindet. Falls mehrere Nachrichtenübertragungen gleichzeitig starten wird ausschließlich die Nachricht mit der höchsten Priorität transportiert. Die restlichen Nachrichten werden anschließend versandt. Aus den oben genannten Gründen fluktuieren die Transportzeiten für den Nachrichtenversand. Die Transportzeit ist die Differenz zwischen der Nachrichteninstanzierung und dem Empfang.

Prozesse müssen teilweise innerhalb einer Zeitschranke terminieren und daher muss Kontrollinformation rechtzeitig zur Verfügung stehen. Darüber hinaus ist ein optimierter mittlerer Datendurchsatz erforderlich da die Leistungsfähigkeit vieler Prozesse von der mittleren Reaktionszeit abhängt. In dieser Diplomarbeit wird eine experimentelle Evaluierungsmethode vorgestellt, welche statistische Daten betreffend des Transportverhaltens ermittelt. Hierfür wurde eine Hardwareplattform entwickelt, welche aus Prozessoren besteht um ECUs zu emulieren. Des weiteren werden auf jedem Prozessor Prozesse simuliert, welche jeweils Nachrichten mit einer eindeutigen Kennung während eines zufälligen und beschränkten Zeitintervalls generieren. Die instanzierten Nachrichten werden mittels CAN verschickt und mit Hilfe der ermittelten Transportzeiten statistische Daten berechnet. Darüber hinaus werden zusätzliche Parameter wie z.B. die Anzahl der versandten und empfangen Nachrichten bei jedem Prozessor bestimmt. Des weiteren wird eine komplette Toolchain vorgestellt, welche zum Entwurf von Testkonfigurationen und Auswertung von ermittelten statistischen Daten dient.

CAN weist akzeptable durchschnittliche Transportzeiten für die Nachrichtenübertragung auf. Jedoch treten sporadisch extremst verzögerte Transportzeiten auf. Die Transportzeiten sind bei einer geringeren Kollisionswahrscheinlichkeit reduziert. Die Kollisionswahrscheinlichkeit wird durch die Anzahl der Nachrichten sowie durch die Sendefrequenz jeder einzelnen Nachricht bestimmt. Die Kollisionswahrscheinlichkeit bestimmt die Auslastung des Transmissionsmedium.

Danksagung

Diese Arbeit entstand am Institut für Technische Informatik, Abteilung Echtzeitsystem, an der Technischen Universiät Wien.

Besonderen Dank richte ich an den Betreuer dieser Diplomarbeit, Prof. Dr. Roman Obermaisser, der mir die Möglichkeit geboten hat dieses ansprechende Projekt zu realisieren. Durch seine Denkanstöße und die Diskussion mit ihm entstanden zahlreiche elegante Lösungsansätze. Darüber hinaus möchte ich mich bei Univ.Ass. Dipl.-Ing. Roland Kammerer für die hilfreiche Unterstützung, Diskussionen und Gegenlesen meiner Diplomarbeit erkenntlich zeigen. Außerdem gilt mein Dank BSc Bernhard Frömel für die Unterstütung bei der Implementierung.

Mein herzlichster Dank gilt meinem Freund Dr. Daniel Ambort für das Kontrolllesen meiner Diplomarbeit und die aufheiternden Gespräche.

Außerdem bedanke ich mich für die konstruktiven Gespräche mit Prof. Dr. Herbert Grünbacher, Dr. Christian Paukovits und Dipl.-Ing. Jakob Lechner.

Ich danke des weitern Ing. Leo Mayerhofer für die technische Projektunterstützung und Maria Ochsenreiter für die administrative Hilfe.

Schließlich danke ich meiner Mutter, die mir dieses Studium ermöglicht hat.

Contents

List of Figures

List of Tables

CHAPTER 1

Introduction

A transmission is the transfer of structured information (data) over a distance. Transmissions are mediated by physical carriers (transmission media) and regulated by subscribers that receive or send transmissions. Furthermore, the transmission media and subscribers are interconnected either directly from subscriber to transmission medium or indirectly via transmission media themselves. One example of a direct interconnection is the Controller Area Network (CAN) bus where the transmission medium is accessed by the CAN standard. CAN bus is used for sending and receiving short real-time messages up to 1 Mbit/s[fS03]. Each message consists of an identifier (id) for tagging and a payload for user data. Furthermore, each message can only be sent if there is no message in transmission. Therefore an error-free message transmission cannot be interrupted. Moreover, messages are related to priorities so that the unique priority (id) of a message determines which message will be transmitted if multiple messages start to transmit simultaneously (prioritization of messages). The prioritization of messages is a main feature of CAN because collided messages (message collisions) will not cause the retransmission at arbitrary later time points. Each collision will be resolved (called arbitration) such that the message with the highest priority will be transmitted and the remaining messages will be transmitted afterwards. Therefore each subscriber has equal communication rights and hence CAN is a multimaster system. The complete CAN standard is explicitly explained in Subsection 3.1.

There are about 800 million CAN Controllers (controller chip that handles the CAN communication) sold per year and the sale volume is still increasing[Zel11]. Furthermore, CAN is mainly used in the car industry[MMTS11]. In addition, the Society of Automotive Engineers Vehicle Network Committee divided auto data transmission networks (a network consists of subscribers which exchange information via physical carriers) into 3 types (A, B and C) according to the SAE J2057 standard[LPL11]. Class A are low speed applications such as a power window actuator, class B are medium speed applications that require fast reaction times and class C are high speed applications. Applications based on CAN are in classes B and C. Furthermore, especially class C networks require predictable transmission times (time difference between message

generation and reception at a receiver) because CAN is used for embedded systems such as a power-train control module. Such a power-train control module has to determine each result in time and therefore it requires information transmitted via CAN in time (time-critical). Therefore the longest transmission time is a main property in a communication system. Each communication system (communication network) consists of independent locations (called Electronic Control Unit (ECU) in an automotive system) that executes processes (processes form together applications) and the processes exchange information via messages. Furthermore, a communication system requires also a minimized average transmission time behaviour because many applications are not time-critical but the performance of the controls depend on the average response times[ZNGSV09]. Each response time is the difference between the time point of an action and the time point to react with such a corresponding action.

The determination of the time behaviour of CAN is difficult for multiple reasons (exemplarily)[PV03]:

- Messages can be generated at different locations (ECUs) without synchronization
- Message collisions
- An error management that is handled by the subscribers themselves (each CAN Controller is responsible for the reduction of its communication privileges if it acts erroneously). The error management influences the transmission behaviour of the erroneous CAN Controller
- Erroneously transmitted messages which delay all queued messages including the erroneous message due to retransmission and transmission of a part of the erroneous message

Furthermore, there are many techniques available to determine the time transmission behaviour of CAN. In Subsection 2.1 a hardware platform was developed to test CAN by explicit test cases so that messages are triggered in time or by events so that message transmissions are explicitly interrupted, etc. The advantages of such a test method are the following: 1) CAN will be tested in real-time and 2) test cases can be explicitly forced. The disadvantage of such a hardware platform is that test cases have to be defined. Therefore only defined test cases are considered and unintended behaviour will not be unaccounted for.

In Subsection 2.2 CAN networks were emulated via simulation software. The advantages of such methods are a fast implementation of a CAN network and that the simulation time advances faster than in real-time. The disadvantages of such methods are that its hard to model the complete CAN standard and that the test method is not fully adequate to a real CAN network.

There are formal methods available which are mainly base on a work from Tindell et al.[TBW95] to determine the longest possible transmission time (worst-case transmission time) for each CAN message (for each identifier). The worst-case transmission time is an important property for applications which must finish within a certain time interval[Kop97]. Such a method is exemplarily presented in Subsection 2.3. The advantages of such formal methods are that the transmission time behaviour is obtained for the assumed worst-case and that any result is obtained rapidly. The disadvantage of such methods is that the worst-case transmission time appears with a very low possibility. Such a possibility of a worst-case transmission time can

be lower than a probability of a hardware failure[ZNGSV09]. Furthermore, it is not possible to fully design the CAN system properties. Therefore many decisions have to be made that will be detrimental or will not meet the bus properties due to of unintended behaviour. In addition, such methods can be determined with more optimistic assumptions that are non-trivial to design. One example of such an approach is presented in Subsection 2.3. Such methods produce more optimistic particular assumptions but may lead to the global misses of deadlines.

There exists also stochastic methods to simulate the transmission time behaviour of CAN which is exemplarily presented in Subsection 2.4. The advantage of such methods is that the CAN network will be simulated by random functions and therefore the average transmission time behaviour will be correctly determined. Furthermore, the simulation clock advances faster than in real-time. The disadvantage of such methods is that the longest transmission times and especially the worst-case transmission times are hard to detect. This is due to particular system components such as processes that trigger messages that are not totally independent from each other.

In general, the state-of-the-art works focus on an artificial approach so that CAN is not evaluated under runtime conditions. Therefore several decisions have to be made for the overall CAN network. Furthermore, A CAN system can be evaluated by simulating particular processes which trigger particular messages at the end of their simulated execution time. In addition, messages can be used to transfer time points which are obtained at the simulated creation time of the message. Moreover, a time point can be determined at the reception of such a message and thus the transmission time can be determined by calculating the difference between the creation time point and the reception time point. Furthermore, processors can be used to simulate the process execution times. Therefore a system consisting of several processors (a CAN Controller is attached to each processor) can be used to simulate a CAN network under runtime conditions. Therefore each processor simulates defined processes and each process is periodically executed. Furthermore, the execution times of a particular process can vary in time. Therefore the system properties vary and thus the transmission times of messages will vary. Furthermore, such a system can be evaluated for a defined period to calculate the average transmission times, variances of transmission times, the longest transmission time, etc. In addition, a particular run that was triggered in order to determine statistical data can be repeated to obtain more statistical data. Therefore repetitive runs that were undertaken in order to determine statistical data have a high significance for the description of the transmission time behaviour of CAN messages.

In a CAN bus system every message is broadcast to all subscribers except the sender (multicast reception). However there are abnormalities that cause an inconsistent reception of CAN messages at the receivers[PV03]. Therefore in such a test system the number of sent and received messages are determined at each subscriber. Furthermore, there are also proprietary star architectures available. A star architecture indirectly interconnects subscribers for message exchange. These prototypes (called CAN star) can be evaluated by the presented technique for their performance. Furthermore, it can also be checked if all subscribers except the sender receive all emitted messages in CAN star network.

This diploma thesis is organized into following chapters:

- Chapter 2: This chapter gives an exemplary overview of the state-of-the-art CAN performance evaluation techniques.
- Chapter 3: It describes the CAN standard as well as two proprietary CAN star architectures. Furthermore, the basic concepts for the prototype implementation to evaluate the performance of CAN are provided.
- Chapter 4: This chapter describes the architecture of the system structure and the experimental process of tests for a CAN based communication system evaluation.
- Chapter 5: Herein the structure of the prototype and the overall test system is described. Furthermore, the toolchains to create the prototype and the configuration data are explained. In addition, the data preparation to show the transmission time behaviour is described.
- Chapter 6: This chapter presents the results.
- Chapter 7: In this section the results and the properties of the prototype are discussed.

CHAPTER 2

Related Work

This chapter gives an overview of the state-of-the-art CAN performance evaluation techniques. Note that the CAN standard is explicit described in Subsection 3.1.

2.1 Multilevel Inspection of Multiple CAN-Networks

Novak[Nov09] presents a framework which consists of hardware blocks that evaluate transmission times, the behaviour in error cases, etc. This framework consists of a set of basic blocks that form a test system for multiple CAN networks. These basic blocks are implemented by reusable units of logic (Intellectual Property (IP) cores - see Subsection 3.4) in a reconfigurable integrated circuit called Field-Programmable Gate Array (FPGA). Each IP core of such a test system accesses and/or monitors CAN buses at the so called Open System Interconnection (OSI) layers 1 and 2. The OSI model divides the communication design into 7 layers such that each layer fulfills a well-defined part[fS96].

Layer 1 interacts with a physical information carrier. Moreover, a layer i is related to layer i+1 but not layer 7 that provides a communication interface for an application such as a monitor function. All layers are stacked according to their numbers such that each result from a layer between layer 1 and 7 is processed by a related layer and forwarded. Therefore it is possible to replace each layer without modifying the remaining layers. Note that only layer 1 and 2 are required for the formation of a minimal communication architecture. Also note that layers may be merged.

In Figure 2.1 the principle layout of the OSI model is shown where two applications at different locations are connected by a physical carrier.

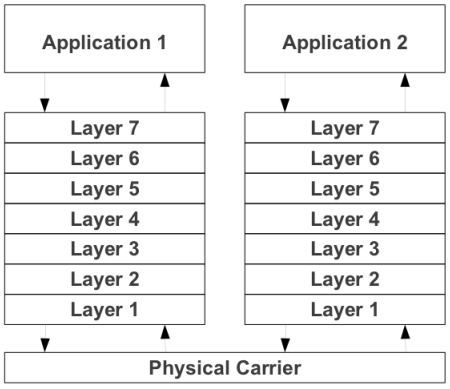


Figure 2.1: OSI model

Layer 1 prepares bit sequences for the specific properties of a physical carrier (e.g. a physical carrier

can be a copper line) and forwards them to the physi-
cal carrier. Layer 1 also receives bit sequences that are emitted at different locations and inverts the preparation for the physical carrier. Layer 2 stacks information (e.g. a message) into a frame (defined as a delimited sequence of sequential emitted bits) and addresses remote stations that are connected to the same physical carrier. In addition, layer 2 also transforms received frames into information suited for layer 3 (e.g. a message). Layer 2 also assures the correct transmission of these frames. The description of the other layers are not defined due to the lack of importance in this diploma thesis. The OSI model is standardized by the International Organization for Standardization (ISO).

The IP cores are related to a global 32-bit time base with a resolution of 1 μs that establishes a global time view as well as compares and captures registers to trigger events. A comparison register stores a number that is constantly compared to the global time base. If there is a match an event will be triggered. Furthermore, the time points of asynchronous events from external buses are stored in capture registers. There are three IP cores (also called IP function) that are used as follows:

- A CAN Controller IP Function that provides the standard functions of a CAN Controller and acts on OSI layer 2. In addition, it provides a transmission queue which stores ordered message objects (a message object is the body of a message i.e. id and payload) for transmission. In addition, each message from such a transmission queue is emitted upon an external request. This request is either hardware-triggered or by the transmission queue and can be described as follows: Each message located in the transmission queue is related to an absolute time point and triggers for sending if a compare register matches the time correlated to the message. Furthermore, received messages are related to their reception times. Upon receipt of a message an output signal is triggered. Moreover, it is possible to force the error state (the error state determines the communication privileges) of the CAN Controller.

- A CAN Trigger IP Function analyzes the traffic at the physical carrier (therefore it acts at OSI layer 1) to provide synchronization. In order to synchronize it is not adequate to detect an already received message due to the need of acting while a message (frame) is in transmission (e.g. to jam a frame that is in transmission). This unit analyzes the bit sequence of a frame and detects patterns within a frame. A detected bit sequence leads to the generation of a trigger signal.

- A CAN Generator IP Function emits predefined frames in such way that the emission of each frame is independent of the requirements of the CAN standard at OSI layer 1 (e.g. it is possible to transmit a complete frame that omitted the arbitration process). Frames are emitted by the description of program sequences. A program sequence defines the time point of the emission of a CAN frame, sequences of frames, build up of the events, etc. Furthermore, it is possible to vary the length of each bit within a frame that is in transmission.

In Figure 2.2 the test system architecture is shown. Furthermore, a test system can observe

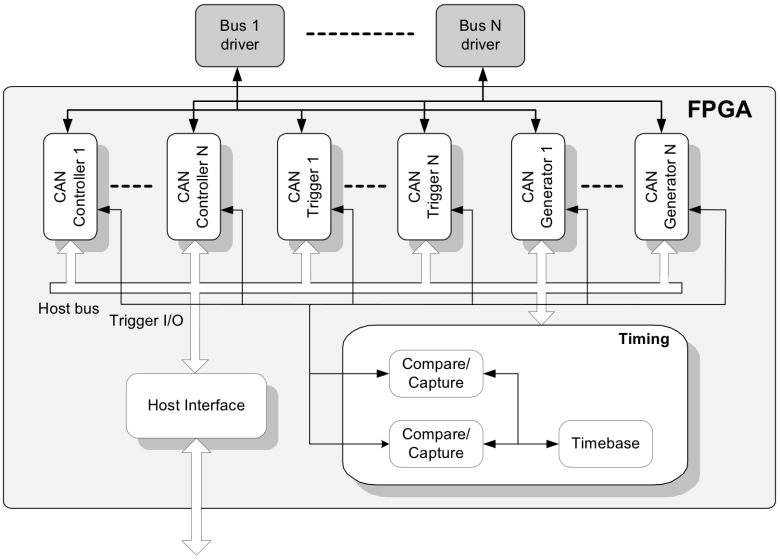


Figure 2.2: System architecture of Multilevel Inspection[Nov09]

1 up to n CAN networks. Each CAN network is connected to at least one CAN Controller IP Function, one CAN Trigger IP Function and one CAN Generator IP Function. All IP cores are interconnected and therefore it is possible that e.g. the CAN Trigger IP Function detects a bit sequence of a CAN frame that is in transmission and triggers the CAN Generator IP Function to jam such a transmission. Furthermore, all IP cores are connected to a host interface that is accessed by an external host. An external host manages the configuration of the IP cores via the host interface.

2.2 Emulation of CAN Networks

Bago et al.[BMP07] evaluate simulation tools which emulate CAN networks for analysis of the utilization of the CAN bus, error detection (transmissions can be erroneous which is traced by the error management), etc. These simulation tools rely on the following parameters:

- Matrix Laboratory (MATLAB) is a mathematical software that can execute matrix operations, draw functions and data, implement algorithms, etc[HC10]. Algorithms are defined by textual description (programming language) placed in a so called m-file and/or higher programming languages such as C. Furthermore, MATLAB can also be extended by packages that provide symbolic computation to provide for example graphical multi-domain simulation.

- Simulink is an extension package for MATLAB that is an environment for multi-domain simulation as well as Model-Based Design for dynamic and embedded systems[Mat].

- Colored Petri Nets consist of place nodes (places), transition nodes (transitions) and directed arcs that connect places to transitions and therefore a Petri net is a graph[JKW07]. The places represent the state of the graph (system) and can be marked by one or more so called tokens. Each token has a data value (token color). Tokens at a certain place are allowed to have a set of token colors (color set of a place). Tokens move from places to connected transitions at other places. This is determined by arcs that connect the nodes. Input arcs of a transition define which token colors are required for a relocation whereas output arcs of a transition describe the modification of the token colors. Therefore a Colored Petri Net characterizes a finite automaton such as a communication system.

TrueTime

TrueTime is a MATLAB/Simulink-based simulator for real-time control systems. TrueTime kernels model nodes that send and/or receive messages. In addition, the behaviour of each node is described by a code function. A code function is written in C++ or a MATLAB m-file and split into segments that are executed sequentially. Each segment consists of an execution code and an execution time that simulates the time consumption of the segment to compute (e.g. a segment simulates the emission of a message and conserves an execution time of 200 μs. Therefore after 200 μs the message is sent and the next code segment starts). Furthermore, TrueTime Network models a network and TrueTime kernels are connected to a TrueTime Network that is within the Simulink environment.

The following CAN bus properties were simulated by TrueTime: prioritization of messages, configuration flexibility (e.g. how easy can a new message be integrated in the system), multimaster and error detection. The test system consists of a single TrueTime kernel connected to a single TrueTime network. There exists sets of messages. Each set is defined by a number of messages with the same send period. Hence there is a code segment for every set and the messages within a set are ordered according to their priorities. In order to add a new node a new TrueTime kernel has to be instanced and a corresponding code function has to be written. Each TrueTime kernel has the same rights for bus access (multimaster). Errors can be simulated by a TrueTime kernel that owns the highest priority message (the highest priority message will simulate errors that cause delays).

TrueTime outputs the busy state of the bus relative to the simulation time and therefore the utilization of the bus can be calculated.

CPN Tools

CPN Tools simulate and analyze Colored Petri Nets (CPN). The colors (properties) of a CPN are described by a textual description called CPN Markup Language. One property of such textual description is time. Each time property prevents one or even more transitions until the simulation time reaches a specific time point. Each transition can be used to e.g. create a single periodic message or to model transmission times.

The following CAN bus properties were simulated by CPN Tools: prioritization of messages, configuration flexibility, multimaster, error detection and automatic retransmission of erroneous transmitted messages as soon as the bus is idle. The simulation consists of a CPN that emulates the bus architecture and that connects via single places to other subnets that represent nodes. Furthermore, each subnetwork that represents a node links to an additional subnet that simulates the creation of messages. The bus simulation CPN models the prioritization of messages at the bus level. In addition, it also forces the prioritization of messages at the subnets and simulates the prioritization of messages at nodes. The bus simulation CPN has a single place where it stores received messages in conjunction with the reception times (communication log) and fulfills the multimaster property. In addition, errors are simulated by a place which blocks the message transition with a defined probability. Furthermore, in case of an error the respective message will be retransmitted. In order to add a new node a new subnet which represents the node as well as the subnet which composes the message set of the node have to be generated and attached to the bus simulation CPN.

From the above mentioned communication log it is possible to calculate the simulation time, message size and utilization of the CAN bus.

2.3 Worst-Case Response Time Analysis

Bit stuffing avoids the occurrence of five consecutive bits with same polarity by inserting bits with opposite polarity (stuff bit). Furthermore, bit stuffing is a CAN inherent system feature. Nolte et al.[NHN03] use a worst-case response time analysis and extend it with a probability distribution for stuff bits which determines the worst-case response time of CAN messages.

Traditional Schedulability Analysis of CAN Frames

The maximum send frequencies of all messages in a network are assumed to determine the worst-case response time for message transmissions. The recurring generation of a particular message can be seen as a traffic stream and therefore the generation of all particular messages is defined as a set S of streams. In analogy to CPU scheduling the set S corresponds to the set of CPU tasks and each $S_i \in S$ is a triple $< P_i, T_i, C_i >$, where P_i is the priority (defined by the message identifier), T_i is the send period and C_i the worst-case transmission time of the frame of a message on the bus of stream S_i. As the minimum variation in queuing time relative to T_i is 0 the worst-case latency R_i for a message i of S_i can be defined by the following criteria:

$$R_i = J_i + q_i + C_i \tag{2.1}$$

where J_i is the maximum variation in the period T_i (relative to the start of T_i) and q_i represents the effective queuing time. The effective queuing time q_i is defined as follows:

$$q_i^n = B_i + \sum_{j \in hp(i)} \left\lceil \frac{q_i^{n-1} + J_j + \tau_{bit}}{T_j} \right\rceil (C_j + 3\tau_{bit}) \tag{2.2}$$

- B_i is the worst-case transmission time of all CAN frames that have a lower priority (in CAN the priority of a frame that transports a message is determined by the priority of the message) than a frame which was sent on S_i. A frame which is in transmission is not interruptible. This implies that a frame that is in a transmission queue has to wait until the bus is idle.
- hp(i) is the set of streams with a higher priority than S_i
- τ_{bit} is the bit time (duration of a single transmitted bit) on the bus that transmits a bit of a frame. The bit time has to be considered due to the different start times of nodes located at different locations reasoned by the propagation delay. The propagation delay is the required time to propagate a signal.
- $3\tau_{bit}$ represents the Intermission (also called Intermission space) defined as the minimum time between frames sent on a bus to transmit messages.

Note that equation 2.2 is a recursion where $(n+1)$th value is calculated by the nth value and q_i^0 is 0.

Equation 2.2 and 2.1 can be rewritten as:

$$R_i^n = J_i + B_i + C_i + \sum_{j \in hp(i)} I_j(R_i^{n-1} - J_i - C_i)(C_j + 3\tau_{bit}) \tag{2.3}$$

$I_j(t)$ is the worst-case number of periodic message creations for a messages j in a time interval of t:

$$I_j(t) = \left\lceil \frac{t + J_j + \tau_{bit}}{T_j} \right\rceil \tag{2.4}$$

J_j is the worst-case creation jitter (difference between shortest and longest delay) in the message period T_j.

Schedulability Analysis of CAN Frames with a Probability Distributions for Stuff Bits

The worst-case transmission time C_i is based on the transmission speed of a bit at the bus, data size, the identifier bits (CAN support two identifier formats with different lengths), overhead bits and stuff bits. The number of stuff bits in the traditional schedulability analysis is assumed for the worst-case and depends on the data transported by the frame in bytes (0 to 8 bytes, denoted as L_i). In addition, it also depends on the identifier format and a fixed number of overhead bits which are adapted for bit stuffing (34 for standard frame format and 54 for extended frame format, denoted as g). As 10 bits are not considered for bit stuffing and τ_{bit} is the worst-case time for bit transport, the resulting worst-case time for a CAN frame transmission according to the traditional approach can be formulated as follows:

$$C_i = (8L_i + g + 10 + \left\lfloor \frac{g + 8L_i - 1}{4} \right\rfloor)\tau_{bit} \tag{2.5}$$

By using a probability distribution for stuff bits instead of the worst-case number a more accurate probability-based response time is yielded. The distribution of stuff bits can be denoted as follows: Υ that is a set of pairs defined as $(x, P(x) \in \Upsilon$ where x is the number of stuff bits and $P(x)$ is the probability of x stuff bits. Furthermore, if multiple frames are sent sequentially and someone assumes that the frames are independent of each other, the resulting joint distribution $\prod_n \Upsilon$ is the combination of n distributions of stuff bits ($\prod_n \Upsilon = \underbrace{\Upsilon \times \Upsilon \times ... \times \Upsilon}_{n}$). Moreover, $\bar{\prod_n}$ is the joint distribution of frames with the same length. The calculation of the joint distribution results in the combination of all probabilities e.g. $(a, P(a)) * (b, P(b)) = (a + b, P(a) * P(b))$ where $a, b \in \{0, 1, ...\}$.

Therefore the transmission time of a single frame on the bus can be expressed as follows:

$$C_i(p) = c_i + \Upsilon_{L_i}(p)\tau_{bit} \tag{2.6}$$

Υ_{L_i} is the distribution of stuff bits in the message and c_i the transmission time of the frame without stuff bits:

$$c_i = (8L_i + g + 10)\tau_{bit} \tag{2.7}$$

Hence the worst-case blocking time by a single frame i is as follows:

$$B_i(p) = b_i + \Upsilon_{\max\limits_{k \in lp(i)}(L_k)}(p)\tau_{bit} \tag{2.8}$$

$\Upsilon_{\max\limits_{k \in lp(i)}(L_k)}$ is the distribution of stuff bits of the blocking frame i and b_i is the worst-case transmission time of the frame i without stuff bits i.e. the longest frame with a priority lower than i:

$$b_i = \max_{k \in lp(i)}(c_k) + 3\tau_{bit} \tag{2.9}$$

Thus the response time of a message from the creation time until reception composes as follows:

$$R_i^n(p) = J_i + b_i + c_i + \sum_{j \in hp(i)} I_j(R_i^{n-1}(p) - J_i - c_i)(c_j + 3\tau_{bit}) + \Psi_i(p)\tau_{bit} \tag{2.10}$$

The distribution of the total number of stuff bits Ψ_i is defined as follows:

$$\Psi_i = \Upsilon_{\max\limits_{k \in lp(i)}(L_k)} \times \Upsilon_{L_i} \times \prod_{j \in hp(i)} \bar{\prod_{I_j(R_i(p) - J_i - c_i}} \Upsilon_{L_i} \tag{2.11}$$

$\Upsilon_{\max\limits_{k \in lp(i)}(L_k)}$ is the distribution of stuff bits of the longest lower priority frame.

Υ_{L_i} is the distribution of stuff bits of the frame of the analyzed message.

$\prod_{j \in hp(i)} \bar{\prod_{I_j(R_i(p) - J_i - c_i}} \Upsilon_{L_i}$ is the distribution of stuff bits of all messages which have a higher priority than the analyzed message.

2.4 Stochastic Analysis

Zeng et al.[ZNGSV09] introduce a stochastic method that determines response times within a distributed system that consists of locally triggered ECUs that communicate with each other using CAN messages. Furthermore, each ECU uses processes (application tasks) to solve defined problems. Messages and application tasks (tasks for short) are mapped to a Directed Acyclic Graph (DAG), where vertexes correspond to tasks and messages and edges describe signals between vertexes. Furthermore, the computations in the DAG periodically proceed in discrete time steps. A time step simulates the behavior of tasks and message transmissions for a time quantum of a real system. Thus a sequence of computations characterizes message transmissions and task executions. Hence this stochastic method determines end-to-end transmission times in a distributed system with clock drifts (the ECUs are not synchronized) and offsets (the ECUs do not start simultaneously).

Basic Definitions

Each ECU has a local clock that triggers computation steps in such a way that tasks are periodically activated, executed and scheduled by their priority. When activated a task reads its input signals and when completed it writes its results into shared variables. A task τ_i can be described by a quintuple (Υ_i, T_i, O_i, $\mathcal{E}_i$, P_i), where Υ_i is the ECU on that τ_i executes, T_i its period, O_i its initial phase, $\mathcal{E}_i$, its execution time and P_i its priority. Each periodic activation of a task is an instance (job) and the jth instance of τ_i is denoted as $\Gamma_{i,j}$ with an arrival time $A_{i,j} = O_i + (j-1)T_i$ which is the time $\Gamma_{i,j}$ is ready for execution.

A middleware acts as the interface between tasks and the CAN bus and supports a special task for this purpose. This task, called transmit task (TxTask), assembles messages periodically from the shared variables and pushes them into a prioritized message queue that is ordered by the id of each message. Messages in a prioritized message queue are transmitted to other ECUs. A message m_i can be described by a sextuple (Υ_i, Υ_i^{SRC}, T_i, O_i, $\mathcal{E}_i$, P_i), where Υ_i is the CAN bus resource used to transmit m_i, Υ_i^{SRC} the emitting ECU of m_i, T_i its send period, O_i its initial phase and $\mathcal{E}_i$ its transmission time. P_i is the CAN id of the message m_i and therefore its priority. The jth instance (job) of m_i is denoted as $M_{i,j}$ with a queuing time $Q_{i,j} = O_i + (j-1)T_i$ which is the time $M_{i,j}$ is ready for transmission.

$\mathcal{E}_i$ is a discrete random variable with a distribution $f_{\mathcal{E}_i}$ that defines the probability for each value within an interval $[E_i^{min}, E_i^{max}]$. This probability distribution is called probability mass function (pmf). In a pmf each time value is a multiple of the granularity τ that represents a time step in a discrete-time model and simulates the computations and communications in a real system. Such computations and communications run on a system resource Υ_k where the hyperperiod H_k is the least common multiple (lcm) of periods of all objects executed on Υ_k. Furthermore, the system resources (ECUs) have independent clocks and therefore the clock difference $\mathcal{O}_{\Upsilon_k,\Upsilon_l}$ describes the clock difference between ECUs Υ_k and Υ_l.

A DAG consists of vertexes V and edges E. V represents the tasks and messages by a set of objects $\{o_1, ..., o_n\}$ and an edge $e_i \in E$ connects two vertexes in the DAG that denotes a data

transfer. Furthermore, a path $\Pi_{i,j}$ is a sequence of objects such that there is an edge between any two consecutive objects. Furthermore, the end-to-end latency $\mathcal{L}_{i,j}$ of path $\Pi_{i,j}$ is the difference between the activation of an instance at o_i and the results produced at o_j.

Task Response Time

The response time of a job depends on the jobs that are queued before. Therefore the P-level backlog at time t $\mathcal{W}_t^P$ is a pmf and defined as the sum of the remaining execution times of all jobs that have a higher priority than P and are not completed until t. The P-level backlog at the beginning of a hyperperiod $G_k^P = \mathcal{W}_{(k-1)H}^P$ is a sequence of random variables $\{\mathcal{G}_1^P, ...\mathcal{G}_k^P, ...\}$ and determined by a Markov chain. A Markov chain M is a triple (S, P, F) where S is a set of states, P is a set of transition function that links all states in S by probabilities such that $\sum_{s' \in S} P(s, s') = 1$ and F is the set of final states in S[AHM05].

After the calculation of $\mathcal{G}^P$, the job releases in order of its release times and modifies iteratively the backlog within the hyperperiod. Right after the release of a job convolution of the backlog pmf with the job execution time pmf ($\mathcal{W}_t^P \star f_{\mathcal{E}_i} = \sum_1^k \mathcal{W}_t^P[m]\mathcal{E}_i[n-m]$) obtains the remaining P-level backlog pmf. Shrinking advances the time from t to t' (time instant right before the next release of a task) by shifting the backlog pmf by $t' - t$ units to the left and by summing up all probabilities defined for non-positive values at the origin. In Figure 2.3 an example for convolution and shrinking is shown.

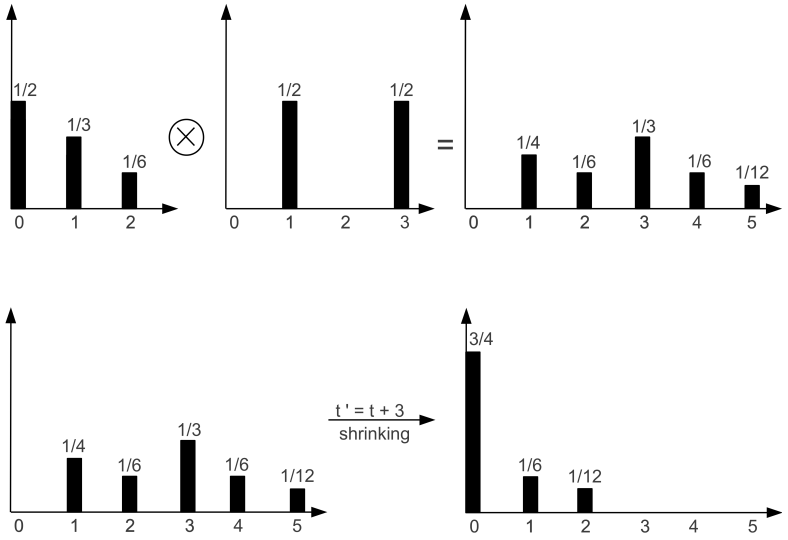


Figure 2.3: Convolution ($\mathcal{W}_t^P \star f_{\mathcal{E}_i}$) and shrinking of $\mathcal{W}_t^P$

The convolution of the backlog pmf with the job response time pmf at the release time of the job yields to the pmf of the job response time. The response time of a job prolongates if a higher priority job arrives during its execution time. Hence the job response time pmf of a job $\Gamma_{i,j}$ is

updated by a split up at the release time of the higher priority job. The convolution of the right hand side with the execution time pmf of the higher priority job generates the actual response time pmf. This is repeated iteratively for all higher priority jobs released after $\Gamma_{i,j}$.

Message Response Times

The P-level backlog W_t^P is a pmf and defined as the sum of all remaining transmission times at time t from queued message instances that are higher in priority than the analyzed one. Furthermore, the pmf of the message instance response time in the approximate system results from its transmission time and the stationary distribution of the backlog at the beginning of the hyperperiod that is adapted for each release of a message instance. In addition, a possible uninterrupted transmission of lower priority job is considered to achieve the pmf of the total transmission time.

Characteristic Interference Message

The interference of messages from remote stations to an inspected message are summarized by a characteristic interference message (characteristic message for short) for each remote station. Therefore the characteristic message represents the load from the remote ECUs. The characteristic message's period T_c is the greatest common divisor (gcd) of the periods of messages at a particular remote ECU that are higher in priority than the analyzed message. In order to obtain the pmf of the characteristic message the following example can be considered. Someone may assume three messages at a remote ECU that have a period that correlates to the transmission time denoted as (period, transmission time): (60,2),(10,1),(20,1). Therefore T_c is 10 and hence there are 6 time instants. At the beginning all messages are queued and so the transmission time is 4 and the transmission times of the following time instants are 1, 2, 1, 2, 1. Thus from a remote message m_i point of view the probability to obtain the interference of 4 transmission time units is 1/6 because there is only on possibility in 6 time instances to obtain this time inference, for 2 transmission time units it is 1/3 and for 1 transmission time units it is 1/2. This can be denoted as $\mathbb{P}(1) = 1/2, \mathbb{P}(2) = 1/3, \mathbb{P}(4) = 1/6$. This example is explained in Figure 2.4. Furthermore,

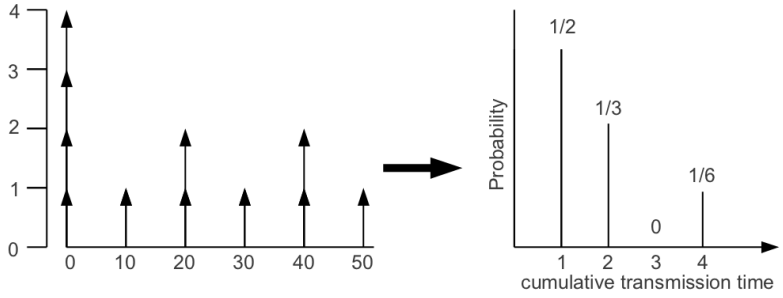


Figure 2.4: Calculation of a PMF for a characteristic message

the characteristic message has a fixed initial offset of $-T_c/2$ and in each hyperperiod a random release jitter in $[0, T_c)$ denoted as $\mathcal{J}_c$. Both are caused by $\mathcal{O}_{\Upsilon_k,\Upsilon_l}$. The higher priority mes-

sages that are located at the same ECU as the analyzed message m_i are not summarized by a characteristic message since they have known queuing instants.

Stationary Backlog Within the Hyperperiod

The backlog W_t^P at the start of the hyperperiod consists of local higher priority messages and characteristic messages. The backlog is adapted in time steps τ. The computation of W_t^P at time t from $t - \tau$ requires an intermediate step arbitrarily close to t and at this intermediate step shrinking advances the time. Furthermore, the possible activation of message instances is determined and the probability of a single activation can be easily computed by $1/n_t$ where n_t is the number of steps from t to the latest possible queuing time. If no new message instance triggers the backlog remains the same and so the intermediate time step transforms to t without modification. Otherwise the transmission time $\mathcal{E}_p$ extends the backlog and so the intermediate time step transforms to t.

Initial Blocking Time

A message that is in transmission can not be interrupted. Therefore even a low priority message can block a message instance $M_{i,j}$. $M_{i,j}$ has a blocking delay $\mathcal{B}_{i,j}$ (also denoted as b) and an instance of a lower priority message $m_{k,l}$ causes a blocking time length $b > 0$ if $m_{k,l}$ has a transmission time $\mathcal{E}_k > b$ and its transmission starts exactly at $\mathcal{E}_k - b$ units before the queuing instant $M_{i,j}$. The probability that such an instance transmission starts exactly at $\mathcal{E}_k - b$ is τ/T_k. Furthermore, by adding up all probabilities of all messages that have a lower priority than $M_{i,j}$ ($lp(P_i)$ denotes the set of all messages with priority lower than m_i) the following formula can be assumed:

$$\mathbb{P}(\mathcal{B}_{i,j} = b) = \sum_{m_k \in lp(P_i)} \frac{\mathbb{P}(\mathcal{E}_k > b)}{\frac{T_k}{\tau}} \tag{2.12}$$

The backlog time is added by convolution to the backlog at the message queuing time.

Message Response Time Calculation

The earliest possible start time for a message instance $m_{i,j}$ is at its queuing time (arrival time) t_q. Furthermore, the backlog computes as described above. An analyzed message instance is queued as long as the backlog at t is not equal to zero. Someone may assume that it takes k steps to obtain a backlog at t of zero. Hence the total queuing time t_k for a message instance can be expressed as follows:

$$t_k = t_q + k\tau \tag{2.13}$$

The total transmission time of a message instance is the sum of the transmission time $\mathcal{E}_i$ and the queuing time t_k.

Stochastic Analysis of End-To-End Latency

In Figure 2.5 an example of an end-to-end delay ($\mathcal{L}_{o_1,o_7}$) is shown. Someone may assume that

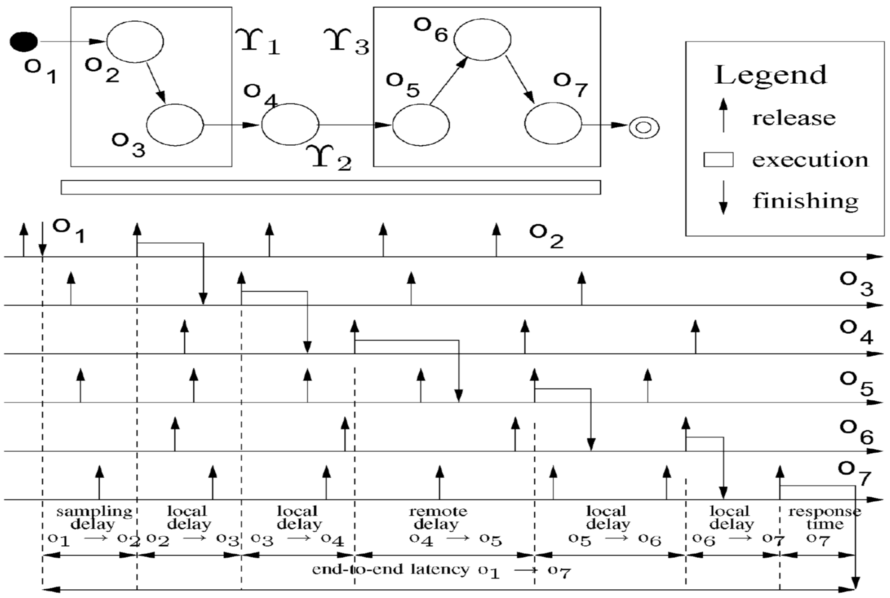


Figure 2.5: End-to-end latency of a path Π_{o_1,o_7}[ZNGSV09]

an external event (modeled as vertex o_1) and a task τ_i (modeled as vertex o_2) that reads the data is generated by this event. Then the time between the occurrence of this event and the activation of the corresponding task that reads the data is called sampling delay. Furthermore, τ_i converts the data from this event to a result and stores it at the middleware. The corresponding task response time is the difference between activation and response as explained in Subsection 2.4. The middleware provides this result to a task τ_j (modeled as vertex o_3) that generates data from the result τ_i and stores them at the middleware. The time between activation of a task that produces a result for a following task and the activation of the following task that reads the result is called local delay. Note that not all results from task instances are part of the end-to-end propagation. The outcome from a job of a task τ_i can be overwritten by the next job of τ_i before recognition at a vertex that follows τ_i. An overwritten value is not propagated and so the delay is not considered for the end-to-end latency.

The TxTask (denoted as o_4) of the middleware assembles the result from τ_j to a message object m_i. The transmission time of this message object is described in Subsection 2.4. Furthermore, a receive task τ_r (denoted as o_5) obtains $m_{i,j}$ and τ_r handles $m_{i,j}$ via the middleware to task τ_k (denoted as as o_6). The time difference between the activation of the TxTask which produced a message instance $m_{i,j}$ and the activation of the receive task that reads $m_{i,j}$ is called remote delay. Furthermore, τ_k computes a result and a task τ_l (denoted as 0_7) receives it via the middleware. The task τ_l stimulates an activator. The time difference between activation by an (external) event and completion (stimulation of an activator) is called response time. The path Π_{o_1,o_7} determines the end-to-end latency $\mathcal{L}_{o_1,o_7}$ by summing up all latencies.

CHAPTER 3

Basic Concepts

This chapter provides the basic concepts for the implementation of the prototype and also describes the CAN standard as well as CAN star implementations.

3.1 Controller Area Network

This subsection describes the CAN Specification 2.0 which consists of Part A that uses a so called standard frame format and Part B that uses the standard and so called extended frame format[Rob91]. In this diploma thesis we will discuss only the CAN standard for Part B.

The subscribers in a CAN based communication system interact by frames to transport messages and each subscriber signals a frame in bits. Each bit that is sent to the bus is characterized by a low state (called recessive bit that represents a logical '1') or a high state (called dominant bit that represents a logical '0'). Furthermore, a dominant bit signalled at the bus is always read as '0' by all subscribers in the system even if a recessive bit is emitted simultaneously. Furthermore, each message that is ready for transmission is transformed into a frame and the transmission of a frame starts if the bus is not in use (bus idle denotes an unoccupied bus). Each subscriber starts the transmission of a frame by emitting a single dominant bit to indicate that the bus is in use and continues by emitting the priority of the frame that is the unique id of the corresponding message. Moreover, each emitting subscriber also determines the value at the bus for each bit that is in transmission. The transmission of the priority of the

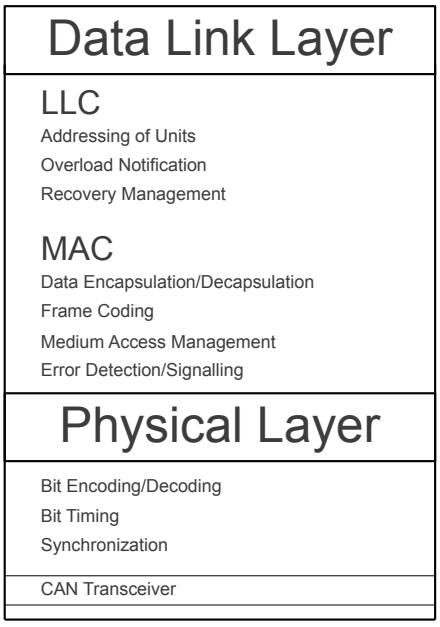


Figure 3.1: CAN Layers

frame (a priority is higher if its number is lower) must
be read back as emitted. Otherwise the corresponding
subscriber may stops the transmission of the frame (an other subscriber sends a frame with a higher priority). The prioritization process that occurs at the emission to ensure the exclusive emission of a single frame is called arbitration. Furthermore, the access technique multiple subscribers apply to check for an idle bus followed by arbitration is called Carrier Sense Multiple Access/Collision Resolution (CSMA/CR).

Figure 3.1 shows the structure of the CAN standard by its layers. Furthermore, the CAN standard by Bosch defines parts of the physical layer (layer 1) as well as the Data Link Layer (DLL) (layer 2) of the OSI reference model. The DLL consists of Medium Access Control (MAC) and the Logical Link Control (LLC). The LLC provides the data transfer as well as remote data request to a node. Each node performs computations and communicates via CAN to other nodes. Moreover, the LLC also manages overload notifications (see Subsection 3.1) and recovery (see Subsection 3.1). Furthermore, the LLC also filters received messages and therefore the LLC is responsible for the addressing of units (see Subsection 3.1).

The MAC provides the data encapsulation/decapsulation (see Subsection 3.1), the frame coding (see Subsection 3.1), medium access management (see Subsection 3.1) and the error detection/signalling (see Subsection 3.1).

The sublayer Physical Signalling of the physical layer provides the bit encoding/decoding, bit timing and synchronization (see Subsection 3.1). The above presented services (DLL and Physical Layer) are provided by a independent unit called CAN Controller to a node.

CAN is defined by ISO 11898[fS03]. The ISO 11898 completely specifies the DLL and the Physical Layer. The Physical Layer consists of the Physical Signalling sublayer, Physical Medium Attachment (PMA) and the Medium Dependent Interface (MDI). The services of the PMA and MDI are provided by a unit called CAN Transceiver. The CAN Transceivers are interconnected by wires and each CAN Transceiver is attached to a CAN Controller. The Subsection 3.1 describes the features of a CAN Transceiver.

The application field of CAN is the car industry but it is also often used in other areas like medical devices, non-industrial machines, marine electronics, factory automation,
etc.[OAFAA06]. For example, a car consists of several ECUs such as engine control units, sensors, anti-skid-systems, etc. and such devices are connected via CAN.

Basic Definitions

This Subsection provides some basic definitions for the CAN standard.

Bus and Bus Level - Recessive/ Dominant

A bus consists of 2 wires: CANH and CANL[Ric02] and these two lines are driven by a CAN Transceiver. Each subscriber that is part of the CAN network is connected to CANH and CANL via a CAN Transceiver. Each subscriber consists of the following parts:

- A node that performs algorithm described by an user.
- A CAN Controller that provide the communication services to the node. The CAN Controller manages the Data Link Layer as well as the Physical Layer.

Figure 3.2 shows the CAN bus with 3 attached subscribers (the two 120 Ω resistors are terminating resistors and their purpose is described in Subsection 3.1).

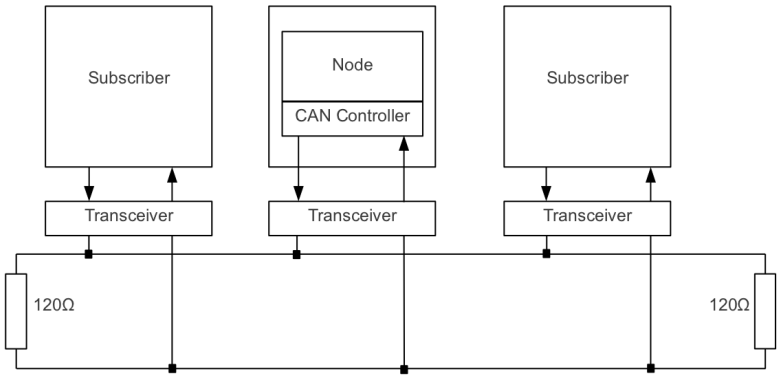


Figure 3.2: CAN Bus with 3 attached subscribers

CANH and CANL are driven by the CAN Transceivers to obtain two different bus levels. The bus level represents the actual physical state of the bus and the differential signalling of CANH and CANL determine the physical state:

- Dominant state (logic '0') if a threshold voltage between CANH and CANL exceeds a defined maximum level. At a dominant state the positive supply voltage drives CANH and the negative supply voltage drives CANL.
- Recessive state (logic '1') if a threshold voltage between CANH and CANL doesn't exceed a defined minimum level. At a recessive state CANH and CANL are driven over a pull-down resistor to ground.

This technique yields to a wired AND-conjunction and so a dominant bus level overwrites a recessive bus level. Furthermore, an idle bus has a permanent recessive state.

Bit Times and Bit Rates

A bit propagated at the bus is a dominant or recessive state kept for a defined time interval at the transmission of a frame and the duration of that time interval (called bit time) determines the bit rate. The bit rate is the number of bits per seconds transmitted and so the relationship between bit time (bit_time) and bit rate (bit_rate) is:

$$bit\_time = 1/bit\_rate \tag{3.1}$$

The bit rate of CAN is up to 1Mbit/s and at a bit rate of e.g. 1Mbit/s the duration of a single bit is 1 μs.

Messages

A message transports information in a fixed format of different but limited length. A message consists of an identifier (id) to definitely tag a message as well as defining its priority (lower number has higher priority) and a payload to transport user data. In CAN there are two messages: Standard messages (to be compatible with Part A controllers) that have an id length of 11 bits and extended messages that have an id length of 29 bits. Both message formats provide a payload length of 0 to 8 bytes.

Each node exchanges information by messages and therefore a node access the CAN Controller and overhands the information to be transported in a message format. Vice versa received messages can be read from a node by accessing the corresponding CAN Controller.

Transmitter and Receiver

A transmitter is a unit that currently emits a message. A receiver is a unit that is not a transmitter if the corresponding bus is not idle. Note that each Transmitter also listens to the bus for e.g. arbitration (see Subsection 3.1).

Frames

CAN provides the following frames: Error frames (see Subsection 3.1), overload frames (see Subsection 3.1), data frames and remote frames (see Subsection 3.1). Data frames transport messages and remote frames request messages. Data and remote frames refers in Part B to a standard frame format and an extended frame format and therefore there exists standard data frames, standard remote frames, extended data frames and extended remote frames. Furthermore, in Part A there are only standard frame formats and CAN Controllers specified by Part B are compatible to Part A CAN Controllers and vice versa as long as only standard identifiers are used.

Data Frames, Remote Frames and Arbitration

Data frames serve as carriers for messages to exchange information between nodes. Furthermore, remote frames are used to request data frames. Each data/remote frame consists of fields and bits and a single transmitted bit of a frame is propagated for a single bit time at the bus. Figure 3.3 shows the extended and standard frame format of a data/remote frame (the bits are emitted from left to right).

Start of Frame

The transmission of a frame starts if the bus is idle and as the bus is idle the access by a subscriber to transmit a data/remote frame is indicated by the so called Start of Frame (SOF). The SOF

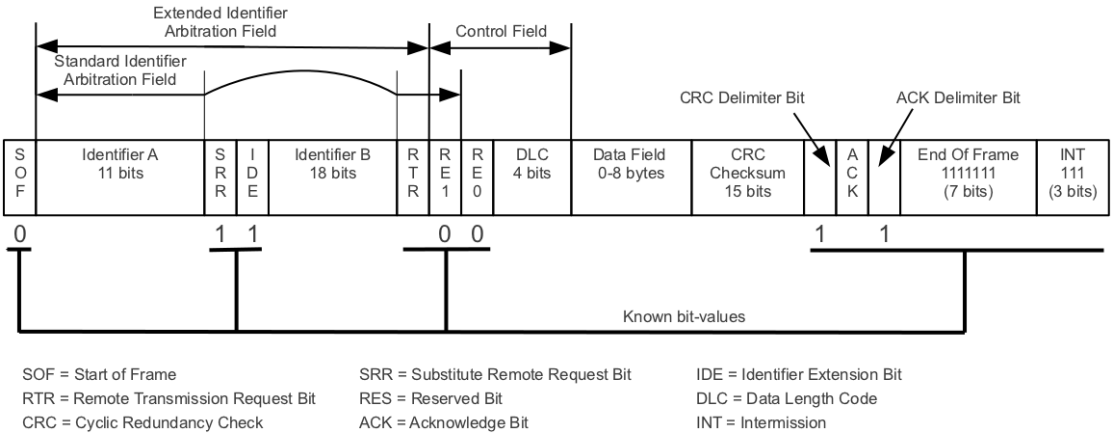


Figure 3.3: Data/Remote Frame

transforms the recessive state of the idle bus into a dominant state for a single bit time and therefore the bus is not idle any more.

Arbitration Field and Arbitration

After the SOF the emission of the Arbitration Field (denoted as Extended Identifier Arbitration Field/Standard Identifier Arbitration Field) continues. The Arbitration Field consists of following parts:

- Standard frames of Identifier A (the identifier of a standard message), the Remote Transmission Request Bit (RTR) and Identifier Extension Bit (IDE). The IDE is merged with the Reserved Bit 1 (RE1).
- Extended messages of Identifier A, the Substitute Remote Request Bit (SRR), IDE, Identifier B (the identifier of an extended identifier is split into Identifier A and Identifier B) and the RTR.

The IDE denotes the transport of a standard frame (dominant) or an extended frame (recessive). An extended frame has the advantage of possessing more identifiers whereas standard frames are compatible to communication devices relayed on Part A and the advantage of having faster transmission due to transport of less bits.

The difference between a remote frame and a data frame is the RTR. If it is set dominant the frame is a data frame and if it is set recessive it is a remote frame. Remote frames are used to request data frames of the same identifier. Moreover, the Substitute Remote Request Bit (SRR) guarantees that the IDE is always a single bit time after Identifier A.

The forwarding of a data/remote frame interrupts if an emitted recessive bit at the Arbitration Field is read back as dominant bit. This technique, called arbitration, prevents the simultaneous

emission of data/remote frames. Arbitration proceeds bitwise and therefore an emission stops directly at the detection of a lost arbitration. Thus an identifier with a lower number has a higher priority

Control Field

After the emission of the Arbitration Field a so called Control Field follows. The Control Field consists of two reserved bits (Reserved Bit 0 and Reserved Bit 1) and a Data Length Code (DLC) that specifies the length of the payload in bytes. The DLC consists of 4 bits (DLC3, DLC2, DLC1 and DLC0) used for numbering and a number greater than 8 specifies a payload of 8 bytes. Furthermore, for remote frames the DLC denotes the payload length of a requested data frame. Table 3.1 denotes the DLC coding for each byte length of the payload ('d' denotes dominant and 'r' denotes recessive).

Number of data in bytes	DLC3	DLC2	DLC1	DLC0
0	d	d	d	d
1	d	d	d	r
2	d	d	r	d
3	d	d	r	r
4	d	r	d	d
5	d	r	d	r
6	d	r	r	d
7	d	r	r	r
8	r	d	d	d

Table 3.1: Length of the payload (data field) denoted by the Data Length Code

The reserved bits are used for further expansion and should be emitted as dominant bits but the receivers accepts them as dominant and recessive. Moreover, the bit Reserved Bit 1 for the standard frame format changed (from Part A) to IDE and therefore the IDE and Reserved Bit 1 are merged.

Data Field

User data are transported in the Data Field. The data field has a length of 0 to 8 bytes for data frames and zero length for remote frames.

CRC Checksum

The CRC (Cyclic Redundancy Check) Checksum is derived by the division of a polynomial. The coefficients of the polynomial are given by the bit stream of SOF, Arbitration Field, Control Field, Data Field (if present), and for the 15 lowest coefficients, by 0. The coefficients are calculated modulo-2 by the generator-polynomial:

$$x^{15} + x^{14} + x^{10} + x^8 + x^7 + x^4 + x^3 + 1 \tag{3.2}$$

The following pseudo code shows the calculation of the CRC Checksum by a shift register (NEXT_BIT denotes the next bit of the bit stream):

Listing 3.1: Algorithm to calculate CRC Checksum

```
CRC_REGISTER = 0;

REPEAT
        CRC_NEXT = NEXT_BIT EXOR CRC_REGISTER(14);
        CRC_REGISTER(14:1) = CRC_REGISTER(13:0);
        CRC_REGISTER(0) = 0;
        If CRC_NEXT then//Division fits at this step
                CRC_REGISTER(14:0) =
                        CRC_REGISTER(14:0) EXOR 0x4599
        ENDIF
UNTIL (CRC Checksum starts or there is an Error condition)
```

Acknowledge Bit, CRC Delimiter Bit and ACK Delimiter Bit

The transmitter emits at the bit time between the recessive CRC Delimiter Bit and the recessive Acknowledge Delimiter Bit (ACK Delimiter Bit) a recessive bit called Acknowledge Bit. The Acknowledge Bit is transformed by any receiver that accepts the transmitted frame to a dominant bit. This process is called acknowledge.

Two recessive bits (CRC Delimiter Bit And ACK Delimiter Bit) surround the ACK slot to obtain a proper delimitation as the ACK bit is emitted by multiple subscribers.

End of Frame and Validation

The end delimitation of each data/remote is achieved by the emission of a sequence of 7 recessive bits. These 7 recessive bits form a field that is called End of Frame (EOF). Furthermore, the receivers and the transmitter accepts a frame or not at the field EOF as follows:

- Transmitter: The frame is valid for the transmitter and therefore the message is transmitted if there is no error until the end of EOF. Each rejected frame has to be retransmitted as fast as possible.
- Receivers: The frame is valid if there is no error until the last but one bit of EOF. Note that it is therefore possible that some receivers accept a frame while others do not if the last bit in the EOF is read different. Thus it is not guaranteed that a messages is broadcast to all receivers. Furthermore, a rejected message is recent and such a recent message causes duplicated messages at some receivers.

Intermission

The Intermission consists of 3 recessive bits. During the Intermission the emission of data or remote frames is forbidden.

Interframe Space

Data and remote frames are separated from preceding frames by a field called Interframe Space. The Interframe Space consists of an Intermission field and an arbitrary long time of bus idle. As data or remote frame are always separated by the Intermission space, it is assumed to be part of a data/remote frame (see Subsection 3.1).

Coding and Bit Stuffing

The emission of each bit is coded by Non Return to Zero (NRZ) and so during each bit time the corresponding bit is either recessive or dominant. Furthermore, after a sequence of 5 continues bits of the same polarity at the emission of the SOF, Arbitration Field, Control Field, Data Field and CRC Checksum, a bit of the complementary polarity is inserted. At the reception of a data or remote frame these inserted bits are removed. This method is called bit stuffing.

Error Frame

An error frame consists of a superposition of error flags and an error delimiter. Furthermore, as a subscriber detects an error it sends an error flag that consists of 6 consecutive dominant bits which breaks the law of bit stuffing. Therefore all other subscribers detect the error flag and also emit an error flag (if not already sent). Thus the emission of an data/remote frame stops immediately. Moreover, all transmitted error flags form the superposition of error flags that has a length from 6 up to 12 bit times. After the emission of the error flag the corresponding subscriber sends recessive bits while reading back dominant bits. Afterwards it starts transmitting seven more recessive bits. Hence an error frame has a length of 14 to 20 bit times.

Overload Frame

An overload frame consists of the superposition of overload flags and an overload delimiter. Furthermore, a subscriber emits an overload flag, consisting of 6 consecutive dominant bits, as follows:

- The internal state of a receiver. A receiver may require extra time to proceed the next data/remote frame. In that case the start of an overload frame is only allowed at the first bit of the expected Intermission.
- The detection of a dominating bit at position one or two during Intermission space. The overload frame starts a single bit time after detection.
- The eight bit of an error or overload delimiter is dominant (a subscriber requires extra time to proceed). The overload frame starts a single bit time after detection. This is not interpreted as an error.

A subscriber that detects a dominant bit at the Interframe Space (position 1 or 2) starts the emission of an overload frame (it detects the overload condition). Therefore all subscribers start the emission of an overflow flag as the Intermission field of a data/remote frame is destroyed.

Note that a single dominant bit at position 3 at the intermission space is interpreted as SOF. Furthermore, after the transmission of an overflow flag the corresponding subscriber emits recessive bits until a recessive bit is read back. At this point the subscriber transmits 7 more recessive bits. Furthermore, at most two overload frames are allowed to delay the next data/remote frame. Note that in a CAN system it is not possible to prevent the emission of a sequence of more than 2 overload frames. Figure 3.4 describes the architecture of an overflow frame.

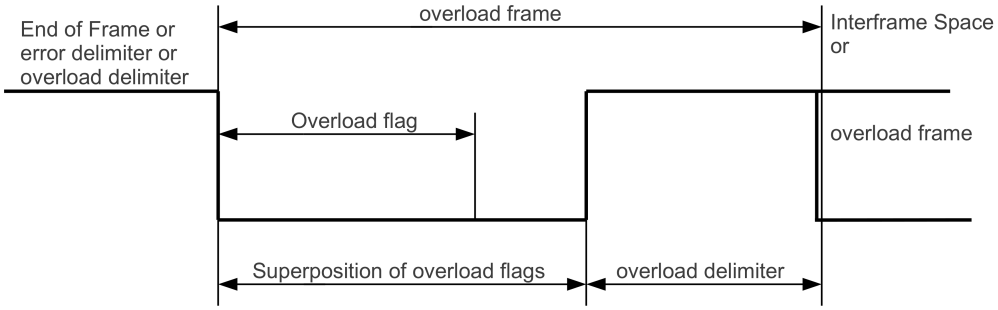


Figure 3.4: Overload Frame

Addressing of Units

Data and remote frames are broadcast to all receivers and at each receiver the frame is converted to its correlated message. All incoming messages are filtered by their identifiers i.e. every bit of a identifier must be either '1' or '0' at a defined position. Furthermore, it is voluntary to mask bits of the identifier such that they are not compared.

Error Management

The error management consists of the detection of an error, the signalling of erroneous frames by each active subscriber and the fault containment for each subscriber. Each subscriber is responsible to detect errors and to report them via error frames to the other subscribers. Furthermore, a subscriber that has a faulty behaviour is responsible to reduce its rights and this is a problematic characteristic of the CAN standard. It is problematic because a faulty behaviour often results from a faulty state of a subscriber and a faulty state may also concern the error containment mechanism. Thus the error prevention and the cause of an error are not independent from each other.

Error Detection

In CAN 5 different types of errors are defined and they are not mutually exclusive:

- Bit Error: Each emitting subscriber also monitors the bus at the current bit time. It is an error if an emitting subscriber reads back a different value than forward but not during arbitration or emission of the ACK slot.

- Stuff Error: It is an error if 6 consecutive equal bits, that are not part of an error/overload frame, are detected while emitting a data/remote frame.

- CRC Error: It is an error if the result of the CRC Checksum in a data/remote frame differs from the result calculated by the receivers.

- Form Error: Some bits in a data/remote frame have a fixed form like the SRR and these bits are called fixed-form bits (see Figure 3.3 - Known bit-values). It is an error if a fixed-form bit has an illegal value but not the last bit in EOF (see Subsection 3.1).

- Acknowledgement Error: It is an error if the transmitter monitors a recessive bit during the ACK slot.

Error Signalling

Each subscriber that detects an error condition signals it by an error frame. A subscriber that detects a Bit Error, a Stuff Error, a Form Error or an Acknowledgement Error start the emission of an error flag a single bit time afterwards. Furthermore, the emission of an error flag caused by a CRC Error starts after the ACK Delimiter if no other error flag has been started before the actual data/remote frame.

Fault Confinement

Each subscriber uses two error counters to determine its communication privileges: Transmit Error Count and Receive Error Count. The Transmit Error Count increases at a subscriber for errors while acting as transmitter and decrements by 1 for successful transmissions. The Receive Error Count increases at a subscriber for errors while acting as receiver and decrements by 1 after a successful reception. The values of these two error counters determines the following three states of an subscriber:

- Error Active: There are no restrictions by the fault confinement. Furthermore, at the start the error counters are reseted to 0 and a subscriber is in the state Error Active if both counters are less than or equal to 127.

- Error Passive: A subscriber in the Error Passive state (error passive subscriber for short) must not send error flags consisting of 6 consecutive dominant bits. Therefore an error passive subscriber that detects an error tries to emit six consecutive recessive bits (called passive error flag). After that process it emits the error delimiter as described in Subsection 3.1. Furthermore, each error passive subscriber has an extended Interframe Space. There is a field called Suspend Transmission that follows the Intermission field. The Suspend Transmission field consists of eight bit times and in that time the error passive subscriber will not send data or remote frames but receives them. Furthermore, a subscriber is in the Error Passive state if one of the error counters equals or exceeds 128 and the Transmit Error Count is not greater than or equal to 256.

- Bus Off: A subscriber that is in the Bus Off state must not influence the bus in any way (e.g. output drivers are switched off). A subscriber is in the Bus Off state if the Transmit Error Count is greater than or equal to 256. A subscriber in the Bus Off state that detects 128 occurrences of 11 consecutive recessive bit transforms into the Error Active state. In such a case both error counters are reseted to 0.

The error counters are increased as described as follows (more than one rule may apply during a given message transfer):

- The Receive Error Count increments by 1 if a receiver detects an error and this error is not a Bit Error caused by the emission of an error flag, consisting of 6 consecutive dominant bits or by an overload flag.
- The Receive Error Count increments by 8 if a receiver detects a dominant bit directly after sending an error flag.
- The Transmit Error Count increments by 8 if a transmitter sends an error flag but there are two exceptions: Firstly for an error passive subscriber that detects an Acknowledgement Error and does not detect dominant bits while emitting its error flag consisting of 6 recessive bits. This is especially demanded during a start-up. During a start-up only a single describer may be online and tries to emit a message. In this case the node only reaches an Error Passive state. Secondly, if a Stuff Error occurs during arbitration caused by a recessive bit that is read back as dominant.
- The Transmit Error Count increments by 8 if a transmitter detects a Bit Error while emitting an overload flag or an error flag consisting of 6 consecutive dominant bits.
- The Receive Error Count increments by 8 if a a receiver detects a Bit Error while emitting an overload flag or an error flag consisting of 6 consecutive dominant bits.
- The Receive Error Count increments by 8 if a receiver detects more than 7 consecutive dominant bits after the emission of its error flag or overload flag.
- The Transmit Error Count increments by 8 if a transmitter detects more than 7 consecutive dominant bits after the emission of its error flag or overload flag.

Physical Layer

The transmission speed as well as physical characteristics and bit representation are defined at the physical layer.

Propagation Delay

CAN uses a shared medium and access it by CSMA/CR. Furthermore, each change of the voltage level at the bus is not instantly propagated to all subscribers. Therefore it must be enough time to establish a common voltage level at the bus e.g. to ensure the correct arbitration process.

The time required for propagation to fulfill this task is called propagation delay.

The propagation delay also limits the maximum network speed for CAN. For example, the bus length can be at most 40m at a bit rate of 1 Mbit/s. 1 Mbit/s is the maximum network speed for CAN.

Bit Time

A physical bit consists of four segments as shown in Figure 3.5. Each segment consists of a defined number of time quanta to determine its length and a time quantum is defined by the number of clock ticks. Table 3.2 explains the usage of these segments. Furthermore, between PHASE_SEG1 and PHASE_SEG2 the bit value is determined and this point for sampling is called Sample Point.

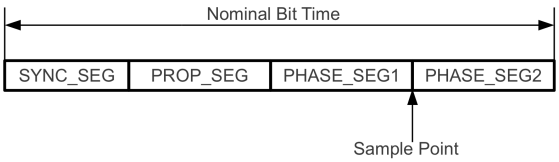


Figure 3.5: CAN: bit time

Name	Propose	Length in time quanta
SYNC_SEG	Synchronization between subscribers. Within this segment an edge is expected	1
PROP_SEG	Compensation of propagation delay	1 - 8
PHASE_SEG1	Stretch bit length (synchronization)	1 - 8
PHASE_SEG2	Reduce bit length (synchronization)	1 - 8

Table 3.2: CAN: bit time

Bit Synchronization

The local clocks of all subscribers start up differently and drift to each other. Hence the subscribers are not synchronized during bus idle. Therefore at the start of each data/remote frame the bit timing is synchronized at each subscriber. This synchronization performs at the borders from recessive to dominant at an idle bus and is called Hard Synchronization. During a Hard Synchronization the internal bit time of each subscriber restarts with SYNC_SEG. Furthermore, the bit timings are resynchronized at each flip of a bit value during the transmission of a frame. Therefore the same technique as for Hard Synchronization is applied if the drift is less than or equal to a defined limit (Resynchronization Jump Width). The Resynchronization Jump Width is at least 1 and bounded by the PHASE_SEG1 but no longer than 4 time quanta. If the drift is larger than the Resynchronization Jump Width the bit length will be modified as follows:

- If the detected edge at the bus is before the Sample Point at the corresponding subscriber, PHASE_SEG1 is elongated by the Resynchronization Jump Width to stretch the bit length.

- If the detected edge at the bus is after the Sample Point at the corresponding subscriber, PHASE_SEG1 is shortened by the Resynchronization Jump Width to reduce the bit length.

CAN Transceiver

This subsection describes the CAN Transceiver characteristics.

Principle Layout of a CAN Transceiver

Figure 3.6 shows the principle layout of a CAN Transceiver. The CAN Transceiver drives two

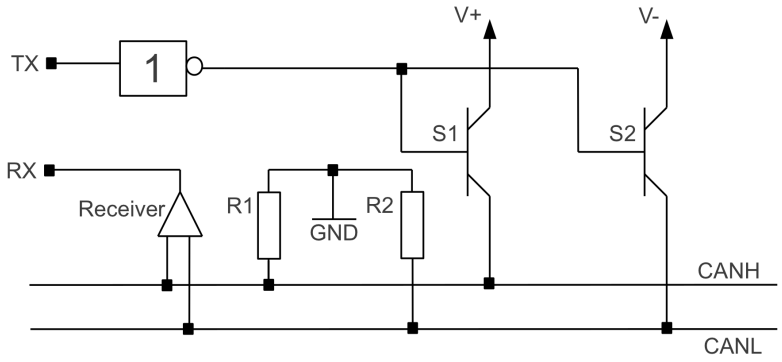


Figure 3.6: Principle layout of a CAN Transceiver

switches (S1 and S2) if a dominant bit shall be transmitted by the TX connector. Thus CANH connects to the positive supply voltage level and CANL connects to the negative supply voltage. Vice versa S1 and S2 are switched off and the two CAN wires are driven to ground by the pull-down resistors R1 and R2. Furthermore, if multiple sources drive two different states, the dominant state is propagated at the bus because the positive and negative supply voltage de-energize at the pull-down resistors. Thus CANH is driven to the positive supply voltage and CANL is driven to the negative supply voltage.

Each CAN Transceiver also reads the bus level by a Receiver unit, shown as 'Receiver' in Figure 3.6. The Receiver compares the voltage level of CANH to CANL. If the voltage level exceeds a defined voltage level the state is read as dominant and the Receiver signals a dominant state ('0') to the RX connector. Moreover, if the voltage level does not exceed another defined voltage level the bit at the bus is read as recessive and the Receiver signals a recessive state ('1') to the RX connector.

Electrical Specifications and Robustness

The ISO 11898 defines electrical characteristics e.g. the Differential Dominant Output Voltage. Table 3.3 shows the parameters that a CAN Transceiver must fulfill.

Parameter	min	max	Unit
DC Voltage on CANH and CANL	-3	+32	V
Transient voltage on CANH and CANL	-150	+100	V
Common Mode Bus Voltage	-2	+7	V
Recessive Output Bus Voltage	+2	+3	V
Recessive Differential Output Voltage	-500	+50	mV
Differential Internal Resistance	10	100	$k\Omega$
Common Mode Input Resistance	5	50	$k\Omega$
Differential Dominant Output Voltage	+1.5	+3	V
Dominant Output Voltage (CANH)	+2.75	+4.50	V
Dominant Output Voltage (CANL)	+0.50	+2.25	V

Table 3.3: Electrical specification of a CAN Transceiver by ISO 11898

The entry 'DC Voltage on CANH and CANL' specifies the voltage level that a CAN Transceiver must resist in the case of short cuts. Furthermore, 'Transient voltage on CANH and CANL' specifies the voltage level that a CAN Transceiver must resist in an error case. Furthermore, the voltage levels for CANL and CANH are assumed for a ground voltage of 2.5 V.

Bus Termination

The CAN bus has at each end a termination as Figure 3.2 shows. This termination is required to reduce signal reflections. Signal reflections disturb the proper propagation of dominant and recessive bits. There are 3 possible terminations:

- Standard termination - A 120 Ω resistor connects at each end of the bus CANH to CANL as shown in Figure 3.2.
- Split termination - The 120 Ω resistors are spitted into two 60 Ω resistors with a bypass capacitor tied between the resistors to ground. Split termination reduces electrical emissions.
- Bias split termination - Same as the split termination with a defined voltage level between these two resistors. Bias split termination set a common recessive voltage level and reduce electrical emissions.

3.2 Star Network

A bus is a single point of failure e.g. a subscriber who permanently emits a dominant level blocks the whole bus for all subscribers (see Subsection 3.1). A star topology is a possible solution for such a single point of failure. There exists active and passive CAN star topologies[BPA09]. A passive star topology does not regenerate incoming signals from its subscribers and thus there are

disadvantages concerning coupling losses, strong limitations on the star radius, etc. Furthermore, there exists active stars that regenerate the incoming signals from the subscribers but do not support fault containment. Thus passive stars and active stars that only regenerate incoming signals have mostly the same limitation concerning single points of failure e.g. broken wires of a CAN bus.

However there are active stars like CANcentrate (see Subsection 3.2) and the CAN router from the Technical University of Vienna (see Subsection 3.2) which support fault containment. CANcentrate acts at the physical and data link layer of CAN and therefore acts like a bus system that is able to block faulty traffic at the OSI layer 2. Furthermore, the CAN router behaves like multiple subscribers that internally distribute received messages to resend them. Therefore this router acts on the OSI layers above 2 and so there are additional services such as routing (e.g. a message is not forwarded to all ports). Routing improves the performance of the global network as the utilization decreases in particular and thus the efficiency of the global bandwidth increases.

However a single star is also a single point of failure but such a CAN star can be placed in a protected area or be built up by high quality components[BRNPA04]. Furthermore, it is possible to replicate a CAN star e.g. ReCANcentrate[BPA09].

CANcentrate

Barranco et al.[BRNPA04] present a CAN star, called CANcentrate, that deals with the CAN traffic from subscribers at the physical and data link layer (see Subsection 3.1). CANcentrate focus on the fault containment at each port and provide its service like a CAN bus with an AND-circuit that interconnects all subscribers. Furthermore, by this AND-circuit the incoming signals from the subscribers are regenerated and forwarded if the corresponding port is not switched off due to errors by emission of a recessive state at the corresponding input of the AND-circuit.

Fault Model

The presented CAN star prevents component faults that may lead to a single point of failure. In particular these faults are:

- Stuck-at node fault: A subscriber may get stuck at a bus level. Therefore a subscriber constantly emits a dominant or recessive state. A permanent dominant state yields to a unusable bus as detailed in Subsection 3.1.
- Shortened medium fault: A bus connected to a battery or to ground by a short cut is deficient. Therefore any communication grinds to a halt.
- Medium partition fault: Each network can split into several subnetworks by a mechanical break of the bus. Therefore the communication of a particular subscriber is not forwarded to all others. In addition, the particular subnetworks may not be accessible at all due to missing termination (see Subsection 3.1).
- Bit-flipping fault: An improperly functional subscriber may emit random bit values and therefore disturb the communication in the network. Such faulty behaviour can occur for

example when a subscriber emits a dominant bit while another subscriber sends a recessive SRR (see Subsection 3.1).

Architecture

Each subscriber connects to the CAN star by two buses to realize a so called uplink to transmit signals from the subscriber to the CAN star and a so called downlink to transmit signals from the CAN star to the subscriber. Therefore two CAN Transceivers are required for each subscriber and thus the uplink is independent of the downlink.

Figure 3.7 shows the principle architecture of the CANcentrate CAN star. CANcentrate con-

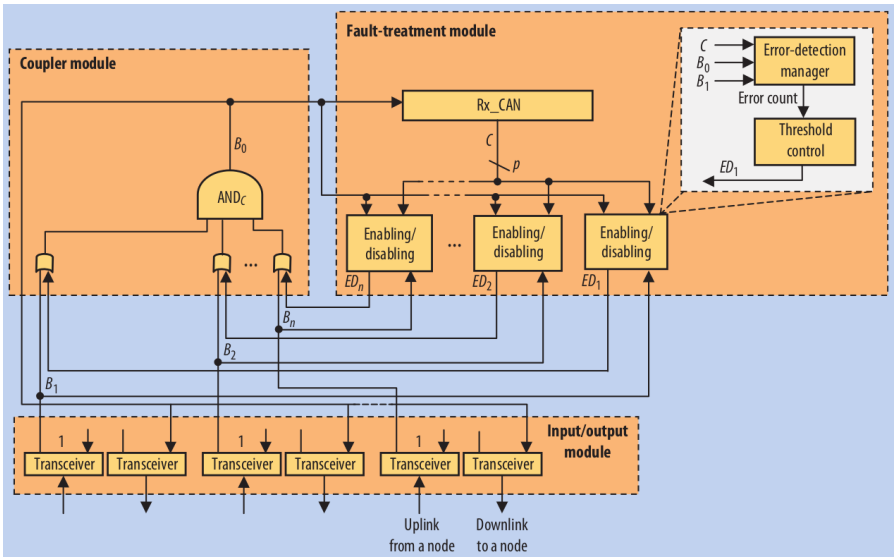


Figure 3.7: CANcentrate architecture[BPA09]

sists of three modules: Coupler module, Fault-treatment module and Input/output module. The Input/output module connects the subscribers to the star coupler by so called ports. Each port consists of an Uplink that connects to the uplink of the subscriber and a Downlink that connects to the downlink of the subscriber.

The Coupler module interconnects the uplinks of the ports by an AND-circuit. An AND-circuit models the AND-conjunction of a bus as explained in Subsection 3.1 and the output of the AND-circuit is called global frame. Furthermore, each port does not directly connect to the AND-conjunction due to an intercalated OR-circuit. The inputs of each OR-circuit consist of the uplink from a port and a connection to the Fault-treatment module such that a port may be deactivated by a logical '1' from the Fault-treatment module. As the Fault-treatment module emits a logical '1' the corresponding subscriber behaves as unconnected due to a logical '1' that is recessive (see Subsection 3.1).

The Fault-treatment module consists of a RX_CAN module and a set of Enabling/disabling units that monitor the behavior of the correlated ports and can either well as to disable or enable

each port separately. The RX_CAN module synchronizes the CAN star with the global frame and generates information corresponding to the global frame.

Fault Detection and Isolation

The faults, described in Section 3.2, are diagnosed by the Fault-treatment module that can either disable or enable ports. Each Enabling/Disabling unit consists of an error-detection manager that can detect errors and a threshold control that can decide the activation/deactivation of a port. The faults at each port are detected and handled as follows:

- Stuck-at node fault: The number of consecutive dominant bits are counted for the uplink and the threshold control deactivates the port if that number exceeds a defined limit T_{nskd}. T_{nskd} is adjustable by the user: $T_{nskd} = (N + 1) * 6$. N defines the number of allowed consecutive error flags (see Subsection 3.1).

- Shortened medium fault: Avoided due to the CAN star architecture.

- Medium partition fault: Avoided due to the CAN star architecture.

- Bit-flipping faults: As mentioned in Subsection 3.1 each CAN subscriber deactivates its services after causing to many errors by its fault confinement and that this fault confinement may be broken. Thus CANcentrate uses the following detection strategies to improve the performance and reliability for bit-flipping faults:

  - If an erroneous contribution of bits occurs at a port and does not result in an error frame at the global frame an error counter called Bit-Flipping Counter (BFC) increases at the error-detection manager for that port. The error-detection manager assumes that the error was not detected by other subscribers.
  - If the error-detection manager detects a global error frame and the corresponding port does not emit an error frame (see Subsection 3.1) the BFC for that port increases.
  - A subscriber that emits an error flag consisting of 6 consecutive dominant bits may detect a Bit-Error (see Subsection 3.1) and therefore restarts the emission of the error flag. This error can only be reasoned by the subscriber it self and therefore the BFC increases if it detects a number of consecutive dominants bits that are not a multiple of the length of an error flag.
  - After the emission of an error flag the corresponding subscriber starts with an error delimiter. The violation of this behaviour leads to an increase of the BFC.

  If the BFC exceeds a defined limit the Threshold Control disables the corresponding port. Furthermore, the BFC decreases for a defined proper performance in particular if the global frame turns to bus idle. The extent of increase and decrease of the BFC has to be defined by the user.

Communication Privileges for each Port

A port starts in a state called idle. The idle state turns into a state called active if the connected subscriber shows up a proper performance e.g. ACK signal (see Subsection 3.1). Furthermore, in the idle state as well as the active state the corresponding subscriber has full communication privileges and the idle state can be used for further improvements of CANcentrate. Furthermore, if a Stuck-at node fault occurs or the BFC exceeds a defined limit the threshold control turns off the port, resets the error-detection manager and therefore the port transforms into a state called disabled. In a disabled state the threshold control monitors the uplink of the corresponding ports and counts the number of times a subscriber sends recessive bits while the global frame is idle. After an idle frame has been counted 127 times the corresponding threshold control transforms the deactivated state into an idle state and therefore the port is active again.

CAN Router of Vienna University of Technology

Kammerer et al.[KOF12] present a CAN star that provides its services as a router. The router provides several subnetworks (a subnetwork is a single bus with attached subscribers) and interconnects these subnetworks. Each subnetwork may introduce faults and therefore failures. The router mediates these failures to the particular subnetwork. Furthermore, received messages are only forwarded to ports defined in tables. Thus the utilization decreases at each subnetworks that do not obtain a message.

Fault Model

In a specific subnetwork a failure (fault/error) may occur. Failure occurrence has to be restricted for the corresponding subnetwork and therefore the router introduces failure modes. The failure modes are in particular:

- Stuck-at node fault
- Crash/Omission Failures: As described in Subsection 3.1 an arbitrary subscriber may not provide its services i.e. sending CAN messages. Thus a subnetwork may not provide a specific service (sending CAN messages).
- Asymmetric Bit-Flip Failures: As described in Subsection 3.1 an atomic broadcast of a message in a CAN bus is not guaranteed if a asymmetric bit-flip occurs at the last bit of the EOF.
- Babbling Idiot Failures: A subscriber may flood the bus with (high priority) messages and therefore disturb/block the communication of other subscribers.
- Masquerading Failures: A subscriber spuriously emits a message with an id that is used by another subscriber located at a different subnetwork.

Architecture

Figure 3.8 shows the router with attached subscribers (CAN nodes). Furthermore, at each port of such a router a CAN bus can be attached via a CAN Transceiver and a particular bus interconnects one or more subscribers. These subscribers, connected to a CAN bus at a port, form a so called CAN segment. Each CAN segment is accessed by a so called CAN Interface Subsystem (CIS) and hence each port is intercalated between a CAN segment and a CIS. Each CIS consists of a CPU, local memory and a CAN Controller. The CPU processes incoming and outcoming traffic e.g. firstly, by receiving CAN messages via the CAN Controller and forwarding them to the other CISes and secondly, by filtering received messages from other CISes as described in the local memory (routing configuration) and by emitting passed messages to its CAN segment. The CISes are interconnected by a Time-Triggered Network-on-Chip (TTNoC) (see Subsection 3.4). The TTNoC conveys traffic in a time-triggered manner i.e. in rounds. Thus there is no internal jitter at the router. Furthermore, each incoming message from a CAN segment is forwarded to the TTNoC. Moreover, the CPU receives messages from the TTNoC. Thus disregarding message transfer each CAN segment is completely separated from the other ports by the CAN Controllers because a particular CPU accesses the CAN Controller and therefore it is not involved in any interaction reasoned by the internal states of the CAN Controllers (see Subsection 3.1).

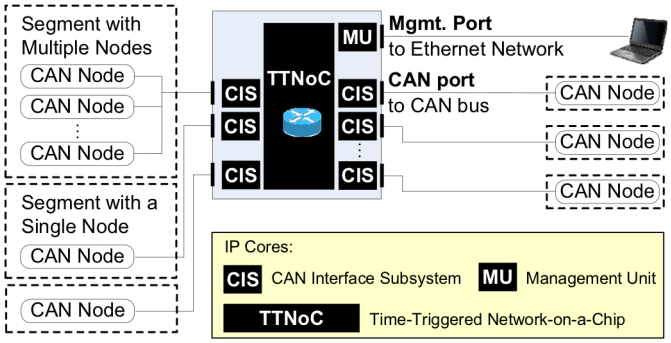


Figure 3.8: Architecture of the CAN router[KOF12]

The time between the reception of messages with the same identifier is measured. If the message interval between two messages with the same identifier is too small the corresponding CPU filters the latest message out. Furthermore, the CPU secures that each specific message is in time i.e. the interval between messages with the same identifier must not exceed a defined time limit. Moreover, the CPU reports unintended behaviour such as a violation of the maximum time between messages with the same identifier via the TTNoC to a unit that is called Management Unit (MU). Therefore the MU has a global view of the system and thus can apply recovery strategies in an error/failure case. Such a recovery strategy may include the reconfiguration of the routing information of each CIS. In addition, the MU can be accessed by a so called management port (Mgmt. Port) for e.g. maintenance.

Fault Detection and Isolation

This Subsection describes how the failure modes defined in Subsection 3.2 are detected and maintained:

- Stuck-at failure: Stuck-at dominant failures are restricted by the composition of the router (star architecture) to the CIS of the particular CAN segment that causes the error. Thus the communication services of the concerned subscribers at the erroneous CAN segment are useless but not the remaining CAN segment because the corresponding CAN Controller (CIS) does not forward stuck-at signals. Note that further consequence will lead to a report of such failures in some way e.g. Messages are not transportable and therefore the CIS will report the management unit of Crash/Omission Failures.

- Crash/Omission Failures: As the CIS secures that each specific message arrives in time and a failure is reported to the management unit. The management unit may apply recovery strategies in such a single case or count such errors over a time interval. Furthermore, if such counts exceed a defined limit the management unit may also apply a recovery strategy. A recovery strategy has to be defined by the user before the beginning of operations. In addition, a recovery strategy is applied during by the reconfiguration of the CISes by the management unit.

- Asymmetric Bit-Flip Failures: A message that was legally received by a CIS will be broadcast to other CAN segments. Furthermore, messages that were retransmitted by a subscriber will be filtered due to the violation of the minimum interval between messages with the same id. Thus buggy atomic broadcasts that happened at a specific CAN segment are not forwarded and thus limited to a CAN segment. However during the emission of a message at the ports this atomic broadcast violation may also happen but they are limited by a specific CAN segment. Moreover, such atomic broadcast violation during the emission can be reduced to zero by limiting the number of attached subscribers to one for each CAN segment.

- Babbling Idiot Failures: If the time interval between the reception of two messages with the same id is too small the corresponding CIS blocks the latest message and reports such violation to the management unit. Thus these errors/failures are limited to the corresponding CAN segment. Therefore each error may be analyzed during maintenance.

- Masquerading Failures: The router has knowledge about the legal CAN identifiers that are emitted at each CAN segment by the subscribers of a CAN segment. Furthermore, the router does not forward illegal CAN identifiers from a CAN segment and reports such violations to the management unit. Thus such errors are limited to a specific CAN segment. Moreover, the impact can be even reduced to zero by using only a single CAN subscriber at each CAN segment.

3.3 Hardware in the Loop

Hardware in the Loop (HIL) is a well known method and used in the test of implemented embedded system[Bac05]. Furthermore, an application for HIL is the real hardware real-time testing[LWFM07]. Therefore a controller model interacts with a real hardware

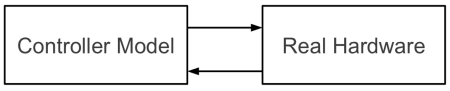


Figure 3.9: Real-time HIL testing

as shown in Figure 3.9. The real hardware can be of any kind as for example a CAN bus. The controller model can be a model of message emissions. Thus the model addresses the hardware and therefore the hardware responses to the model. Furthermore, the received data from the hardware may be used to trigger the hardware more frequently. However such a test run for some defined time and the test setup strongly depends on the application to be simulated.

An Example of HIL

Fennibay et al.[FYS10] present a HIL method to evaluate Hardware/Software Co-design systems. A Hardware/Software Co-design system consists of hardware and software. Furthermore, the overall system consists of subsystems and each subsystem is either a virtual system or a real system. These subsystems communicate with each other via channels. Thus real systems can be combined with system models to evaluate one or several real subsystems and/or one or more modeled subsystems in a loop.

SystemC

SystemC is a C++ framework and consists of a simulation kernel and a class library[BvL11]. The library consists of classes, macros and templates in order to model concurrent systems. Furthermore, each SystemC model consists of modules and each module represents an encapsulated part of a system. Modules exchange informations by channels and consist of processes. The processes of a module define its behaviour. Each process is based on a C++ member function. Thus SystemC models Hardware as well as Software.

SystemC simulation kernel is a discrete event simulator[GLMS02]. The event simulator consists of an event queue, event handler and a state. The state represents the actual status of a simulated system and changes during events. Each event occurs at a discrete time point and is queued and ordered by a related time stamp (a time stamp represents such a discrete time point) in the event queue. Furthermore, the event handler processes events to proceed changes at the state and add/remove events to/from the event queue. Thus the simulation time proceeds due to the time stamps of the events.

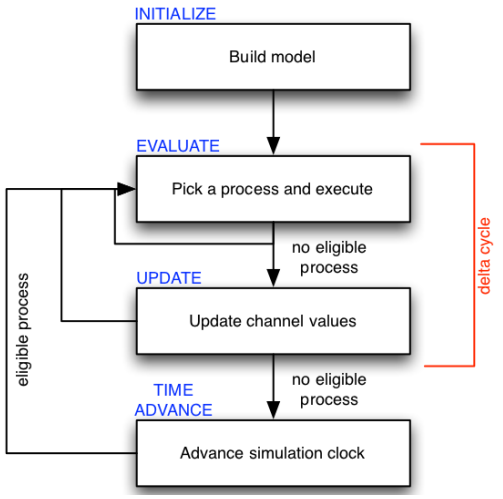


Figure 3.10: Flow of the SystemC scheduler[FYS10]

The performance of a SystemC kernel can be divided into four main phases: 1) initialize, 2) evaluate, 3) update and 4) time advance as shown in Figure 3.10. The evaluate and update phases

form a delta-cycle which represents an infinitesimal time amount to pass. In a delta cycle concurrent operations are performed in the evaluate phase and as soon as these concurrent operations complete the update phase continues. This process persists as long as sequential performance of concurrent operations are not completed yet. Furthermore, after the completion of all operations the simulation advances in time and therefore continues with the next time stamp from the event queue.

SystemC simulation kernel has no external interface for external events i.e. new data arriving from outside will be handled at the next time stamp in the event queue even if they trigger events. For example, someone may assume that the event queue contains the following events with timestamps 00:02 and 00:06, respectively, and the data arrives at 00:04. In this case the model will correct the simulation clock directly from 00:02 to 00:06 and thus the data will be received at 00:06.

Architecture

Figure 3.11 shows the architecture of the HIL with SystemC. It consists of virtual subsystems

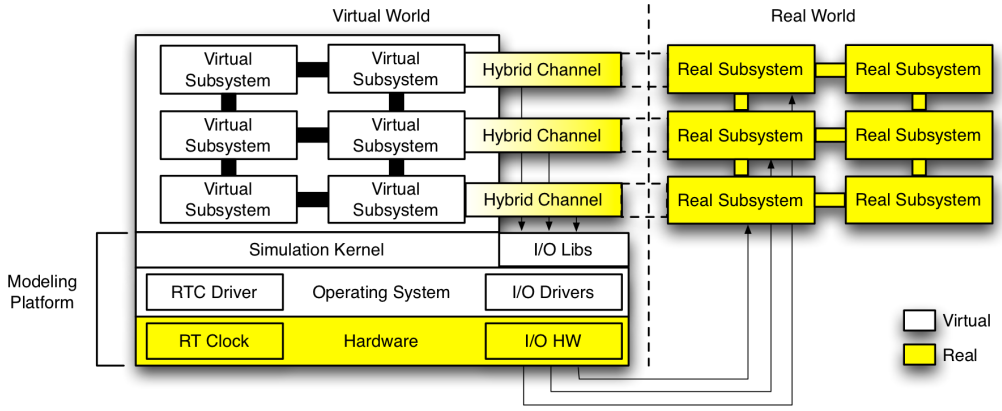


Figure 3.11: Architecture of a HIL and HW/SW co-design for real-time embedded systems[FYS10]

and real subsystems. Each real subsystem can be any sort of an implemented system such as for example a building automation station. The real subsystems can interact by an arbitrary existent communication technique.

Furthermore, the virtual subsystems are modeled by SystemC and run on a simulation kernel. The simulation kernel runs on general-purpose operating system which operates on a computer hardware. The computer hardware plus the general-purpose operating system and the simulation kernel form together the modeling platform. The modeling platform is trimmed to reduce jitter and latency and therefore the e.g. power management and swap memory are disabled. Furthermore, the virtual subsystems communicate with the real subsystem by so called hybrid channels. These hybrid channels abstract the communication of the I/O hardware which can be a Universal

Serial Bus (USB).

The hybrid channels are an extension of the SystemC channels and can carry digital or analog data. Firstly, the data can be transferred from a virtual subsystem to a real subsystem in different phases as follows:

- Evaluate phase: Produced data are instantly transfered.
- Update phase : Produced data are transferred after the completion of a delta cycle.
- Time advance phase: Produced data are transfered after the completion of all delta cycles.

Secondly, the data transfer from a real subsystem to a virtual subsystem is also characterized by the following factors: There is a polling mechanism implemented in the system that periodically checks an external interface for arrived traffic. Thus arrived data are forwarded to the SystemC kernel with a small amount of latency.

3.4 System on Chip

A Multiprocessor System-on-Chip (MPSoC) (see Subsection 3.4) is a System on Chip (SoC) (see Subsection 3.4) that contains multiple processors. A SoC is a complete system architecture placed onto a single chip and not onto many separated subsystems. A SoC can be placed onto a reconfigurable hardware such as a FPGA (see Subsection 3.4) and has a high system complexity. In order to reduce complexity so called IP cores that require well-defined interfaces (see Subsection 3.4) were incorporated into the SoC. Each IP core is a functional unit e.g. a processor. Furthermore, these IP cores are linked by interconnects such as the Altera Avalon Interface (see Subsection 3.4) or by a Network-on-a-Chip (NoC) (see Subsection 3.4). An interconnect system is best suited for direct connections whereas a NoC for a message based network topology. Such a network topology can be further extended with features e.g. fault isolation. A NoC together with its extended features and an attached component forms a system. This system can be based on a time-triggered architecture. An example of such a time-triggered system is the Time-Triggered System-on-Chip (TTSoC) (see Subsection 3.4)

Basic Definitions

This Subsection provides basic definitions as well as basic descriptions for SoCs.

System on Chip

Modern chips are highly integrated and still become more complex due to Moore's law[Mol06]. Furthermore, divide and conquer is a method for simplifying complex systems i.e. to split a system into modules. The modern state-of-the-art chip contains a multi-chip System on Board (SoB) that supports all entire services such as digital logic[SWM+06]. Thus a SoC is a chip consisting of blocks to offer all services of a complete system. The main advantages of a SoC are flexibility (buildup of system components), (re)programmability and module re-use[OMFK08].

Intellectual Property Core

An IP core is a reusable unit of logic e.g. a processors[KB02]. Furthermore, such IP cores can be used in a SoC to design a system. A system, based on IP core technology, consists of modules (IP cores) that communicate via an interconnect such as the Altera Avalon Interface (see Subsection 3.4) or a NoC. Furthermore, the usage of IP cores yields to a plug and play integration as each IP core provides a high level of abstraction and an exact and independent interface specification[Alt11a]. Moreover, the design process consists of three stages: 1) specification and documentation, 2) implementation and 3) verification.

On-Chip Networks

A SoC that consists of separated blocks such as IP cores demands a logical connection to enable communication between them[HWC04]. This logical connection demands a physical wiring and can be realized by a bus system. Furthermore, a well-defined communication architecture leads to a clean separation of the connected subsystems. Thus each subsystem should interact with the bus system by messages in order to separate data and control flow. A NoC transports messages between connected communication partners. Furthermore, he wiring should be a mesh, like in Figure 3.14, due to the optimization of the total jitter of every possible communication in the network and the predictability of the transport times.

Multiprocessor and Reconfigurable System-on-Chip

A MPSoC is a SoC consisting of multiple programmable processors and each processor is a system component[WWM08]. Each processor has a special purpose to fulfill unique requirements of an embedded system and thus an application-driven design approach hardly influences the system architecture. Hence standards have a key function because they yield to heterogeneous systems which share definitions of applications. Furthermore, standards require a well-defined interface specification that also reduces the effort to split complex and resource demanding programs on separate chips. Therefore a proper system design process derives and so multiple and independent designers can act at random systems. Moreover, a weak design concept and poor interface specification will yield to improvements of low impact and work steps that impair the system performance in some ways. Thus a poorly conceived system design leads to a greater chip area that profoundly affects the power consumption. Vice versa the chip area will scale down and therefore a merged system reduces the power consumption dramatically[HWY+11].

A single threaded and sometimes complex reference implementation is often provided for standards. Therefore such reference implementation can be oversized. Oversized software leads to poorly conceived parallelization, time-consuming optimizations for hardware and increased power consumption. Thus such standards have to be adopted for a MPSoC and optimized for power and performance. Furthermore, a well-defined MPSoC platform supports different specialized processors which can be easily maintained, programmed and configured. Therefore such systems meet the demands of an application much better than a general-purpose system. Moreover, Hardware-Software Codesign is important because a processor in a SoC can be ad-

justed by extending its architecture or by using IP cores that provide complex operations within a few computational steps. Reconfigurable hardware platforms (FPGAs) play a key role in the MPSoC design[SBR+07]. Reconfigurable hardware provides the possibility for proper separation of functional blocks such as e.g. IP cores and for early integration of these functional blocks. Reconfigurable hardware can be efficiently validated as the integration of functional blocks can be fulfilled in small steps. Each step can be validated and therefore each validation is limited and reduced in complexity. Furthermore, the overall system can be adjusted during the development process. System adjustment in small steps provides various possibilities for hardware and software redesign or for removal of analyzing untis after the design process.

Avalon Interconnect

The Altera Avalon Interconnect interlinks IP cores or couples an IP core to the environment[Alt11a]. There exists the following seven interfaces for different purposes:

- Avalon Streaming Interface that is unidirectional and used for high bandwidth data transfers.
- Avalon Memory Mapped Interface that is a memory mapped master-slave system. Each memory mapped master-slave system consists of a master-component that access and control one ore more slave-components by addressing memory. Furthermore, a memory mapped interface is used in order to tightly couple hardware.
- Avalon Conduit Interface for connections that do not support the Avalon Interconnect standard.
- Avalon Tri-State Conduit Interface to connect off-chip resources.
- Avalon Interrupt Interface to provide event notifications (Interrupts).
- Avalon Clock Interface that provides or drives a system clock.
- Avalon Reset Interface that provides a reset signal.

An IP core can be connected with one or more Avalon Interfaces to other IP cores or to the environment. Furthermore, each control signal can be logical low or logical high driven i.e. a high state ('1') or a low state ('0').

Clock and Reset Interface

An IP core typically receives a system clock that defines the duration of a single cycle. During each cycle the digital logic of an IP core has to make a decision such that the clock will provide synchronization for the internal logic. For example, a single instruction of a processor placed on a SoC takes one cycle (computational step) to complete. Thus a single hardware operation takes one cycle of the given clock signal. Furthermore, a reset lane indicates a non-reset or reset request that transforms the internal state of an IP core into a defined ground state.

Memory Mapped Interface

A memory mapped master/slave communication couples tightly hardware for easy and fast control that can be used to e.g. couple memory to a processor. Furthermore, each slave requires a connection to at least one master and a connection consists of at least a chipselect signal, a read and/or write buses, a read and/or a write-signal and address lanes. Address lanes are required to form a binary address to control a slave for e.g. address a memory area. An addressed memory area can be used to read or write data and these data have to be transported by a read bus that is used for reading and a write bus for writing. Furthermore, the write signal and read signal decide the mode of operation e.g. the decision between writing (enabled write signal) and reading (enables read signal). Furthermore, a master activates a particularly attached slave by the chip-select lane of the slave to perform operations (reading or writing). Note that more interface signals are described in the Altera Avalon manual.

Streaming Interface

A streaming Interface is not coupled to a system clock and used for transferring huge data amount. Typical applications are multiplexed channels or packets which are send unidirectionally in blocks. Figure 3.12 shows the layout of the Altera Streaming Interface.

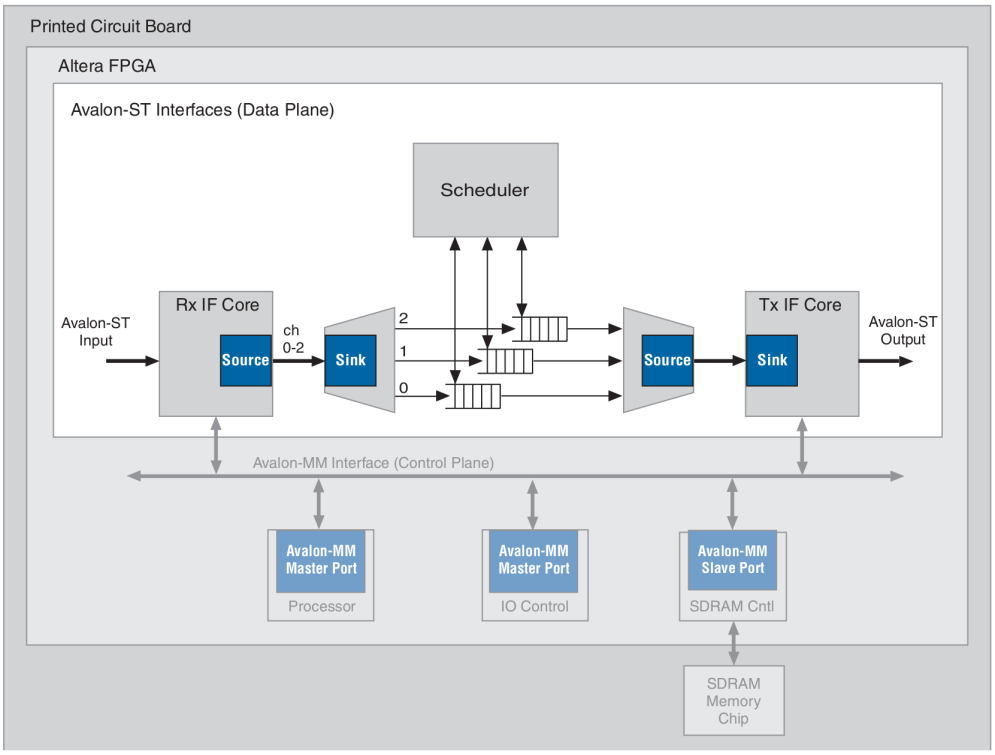


Figure 3.12: Layout of Altera Streaming Interface[Alt11a]

The Streaming Interface supports sideband signalling to indicate errors, backpressure to regulate the data throughput at a receiver, start and end of packet delineation, data bursting and multiple channels. Figure 3.12 shows 3 channels from a source interface that are managed by a scheduler to multiplex them to a receiver interface. Therefore the channel lanes signal the channel number. Moreover, a ready signal indicates backpressure from a receiver to a sender to control the intermissions for reducing the data throughput. In addition, a valid signal subscribes the integrity of transmitted data. Moreover, error lanes form a code word to message errors to the receiver. Furthermore, the use of a packet based system requires a signal to mark the start of a packet, a signal to inform the receiver of the end of a packet and a lane to identify empty symbols in a packet.

A streaming interface is not suitable to transfer safety critical data because this interface is not well-defined for multiple receivers. There is a mess between control and data information and jitter. In general a streaming interface for multiple receivers results in high costs for the chip area.

Interrupt Interface

Slaves (senders) can use an interrupt interface to indicate urgent events to a master (receiver) e.g. a timer raises an interrupt for a processor to signal an overrun of its counter. Such a signal from an interrupt interface is called Interrupt Request (IRQ). An IRQ, raised by a sender, has to be asserted until a receiver handles a routine. This routine consists of accessing the slave by a memory mapped interface to read the interrupt status register of the sender, some possible additional steps depending on the application and the status of the interrupt service register and acknowledge. Furthermore, a sender uses a single interrupt signal to connect to a receiver that can be connected to multiple interrupt lanes from multiple slaves.

Conduit and Tri-State Conduit Interface

A conduit interface groups inputs, outputs and bidirectional signals which are not designed for the above mentioned interfaces. The applications for the conduit interface are consisting of tightly coupled hardware and make the connection to non-Altera-Interfaces possible.

A Tri-State Conduit interface drives off-chip components. Furthermore, the pin consumption can be reduced by sharing data pins, address pins and control pins. In addition, the pin consumption can be reduced by multiplexing off-chip elements and docking two or more master nodes to an interface. Moreover, it is possible to tri-state a pin to switch off a physical connection to an off-chip unit due to circuit requirements and multiplexing.

Time-Triggered System-on-Chip

The complexity of a system can be reduced by a high level abstraction[Kop08]. A high level abstraction can be defined as a network that is accessed by its attached components. Thus it is possible to place a complete system onto a single chip by putting a network and attaching components. Furthermore, a proper delimitation between the components and the network is required to increase the system reliability as well as the abstraction. Therefore a time-triggered approach

that is a Time Division Multiplex Access (TDMA) technique can be used for this purpose. The TDMA technique is a method to share a single network for multiple components[Kop97]. Each component has a defined time quantum within a periodically repeated interval to use the single shared network. Furthermore, the TTSoC is such a time-triggered network with a proper delimitation between components and network. It consists of an network (TTNoC), an interface between each component and the network (TISS) and resource management (Resource Management Authority and Trusted Network Authority)[Pau08]. Figure 3.13 shows the architecture of the TTSoC.

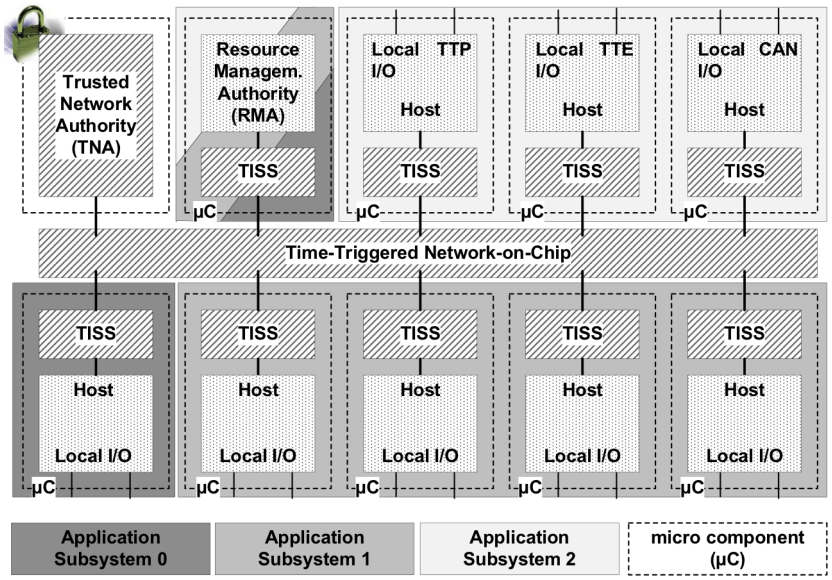
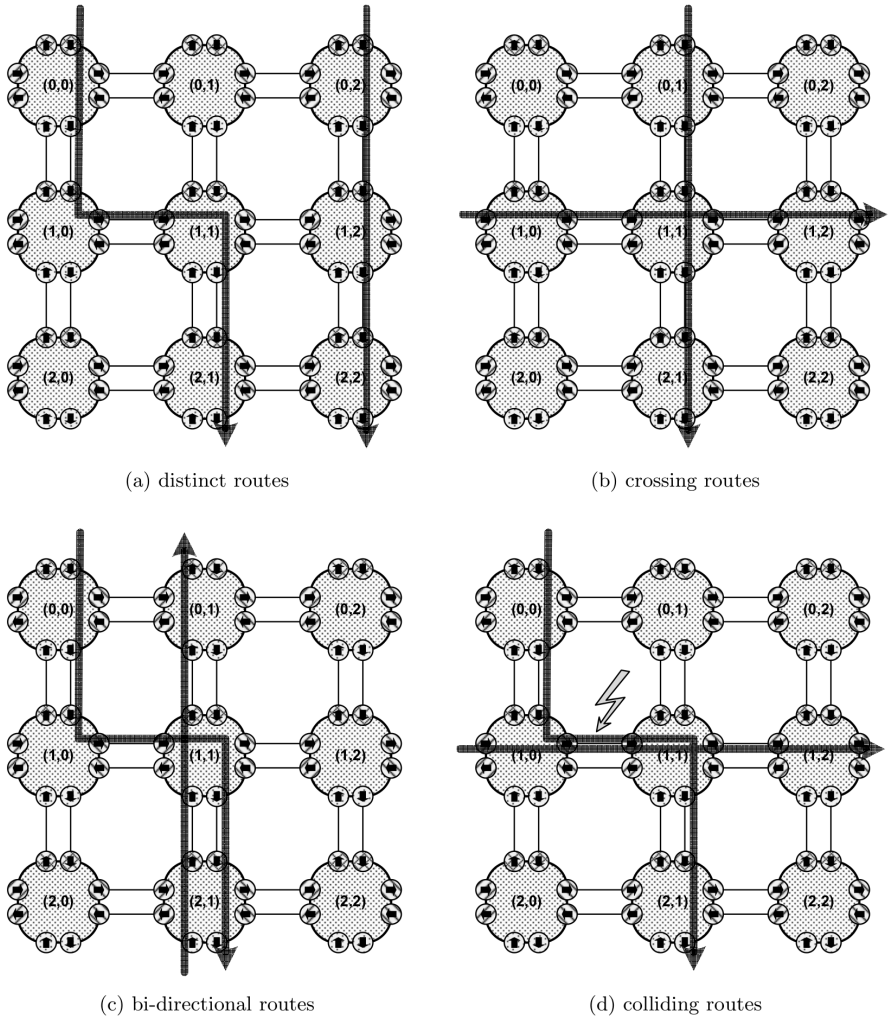


Figure 3.13: Structure of the TTSoC architecture[Pau08]

TTNoC

The TTNoC routes stateless encapsulated channels from one endpoint to another. Furthermore, the sender and the TNA (see Subsection 3.4) manage the state of the communication. Moreover, the TTNoC consists of fragment switches consisting of four directions and which are placed in a mesh. Each direction has a lane for outgoing data (32 bits wide) and a lane for incoming data (32 bits wide), a valid line to separate data frames and a header line to indicate routing information. Furthermore, a frame consists of flits and each flit is a data burst that is transmitted within a single clock cycle. Figure 3.14 shows a TTNoC architecture consisting of fragment switches.

In Figure 3.14 'a' there is a route from the top left fragment switch to the bottom middle fragment switch. Each fragment switch requires control to connect a particular input to a particular output. Thus a fragment switch requires routing information to direct user data. The routing information is conveyed by flits. Each flit that contains routing information is indicated by the corresponding header line. Therefore the header lines is assigned to true at the corresponding fragment switch. The fragment switch configures by taking the first 4 bits of the incoming data



(a) distinct routes

(b) crossing routes

(c) bi-directional routes

(d) colliding routes

Figure 3.14: Simultaneous routes in a network-on-chip[Pau08]

lane and each of these 4 bits substitutes a direction to north, south, west and east. Thus the port that received the routing information connects to the outcoming ports which are indicated by the 4 bits e.g. the fragment switch located in the middle in Figure 3.14 'a' switches the incoming data lane of the west direction to the outcoming data lane of the south direction when receiving the routing information. Furthermore, after switching a particular fragment switch, the corresponding routing information is shifted by 4 places to the left and forwarded to the following fragment switch.

It is possible to route more than one channel through the network as well as multicasting as long as a particular data lane is only used once. Figure 3.14 shows possible routing of 2 channels at the same time.

Trusted Interface Subsystem and Micro Component

A Trusted Interface Subsystem (TISS) is a network interface to connect an IP block to the TTNoC for providing core services e.g. time stamps or a communication interface. Furthermore, the TISS guarantees the isolation of broken IP blocks by a TDMA technique. Moreover, the TISS controls the fragment switching as described in Subsection 3.4 as a sender.

A TISS and a connected IP block form together a so called Micro Component. It is important to mention that a TISS can also connect the RMA (see Subsection 3.4) to the TTNoC. Furthermore, each Micro Component is an independent system that may communicate with other Micro Components or Input/Output (I/O) to other systems.

Resource Management Authority and Trusted Network Authority

The Resource Management Authority (RMA) manages the resource allocation for the applications located on Micro Components as well as the communication system e.g. reserve or cancel encapsulated channels or inform a TISS of changed application constraints. Furthermore, Micro Components access the RMA that is only allowed to change system configurations by accreditation of the Trusted Network Authority (TNA).

The TNA audit changes made by the RMA and manages the TTNoC. Furthermore, the TNA can not be accessed by any Micro Component.

Global Time Base

The TTSoC provides a synchronized global time base that is a sparse time base i.e. a time stamp of a particular TISS is comparable to any time stamp of any TISS. The sparse time base is characterized by the mapping of events to time points. Events are mapped to so called active time intervals and any two active time intervals are separated by intervals of silence[Kop97]. Any event occurring within an interval of silence has to be assigned to the neighboring active time intervals.

The current implementation of the TTSoC architecture provides 64 bit wide time stamps, based on the global time base. Furthermore, the time stamps are obtained by accessing the TISS. Each TISS support the 64-bit time format by two 32 bit words. Furthermore, the accuracy is up to 465.66 ps.

CHAPTER 4

System Model

This chapter describes the architecture of the system structure as well as the experimental process of tests for a CAN based communication system evaluation.

4.1 System Structure

A subscriber in addition to a physical clock that triggers messages for sending communicates by transmissions with other subscribers. The test system consists of such units that are interconnected via a network and linked to a consistent global time base for measuring transmission times. Figure 4.1 shows the structure of the test system.

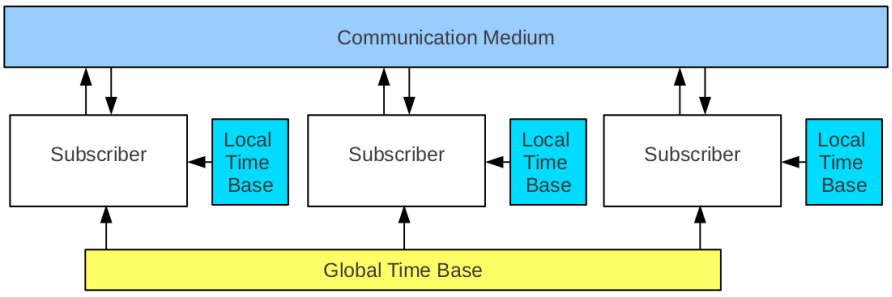


Figure 4.1: Structure of the test platform

Time Bases

A signal is the change of a magnitude over time and a periodic signal is a signal that is recurring in time intervals. A physical oscillator is an instrument that generates a periodic signal that is called clock signal and a physical clock consists of a physical oscillator and a counter that is incremented at the end of the period of the clock signal. If the counter of a physical clock is

incremented it is called microtick[Kop97]. Therefore a physical clock is a device for measuring time in microticks.

A physical clock that is not synchronized with other physical clocks and locally used (i.e. at a subscriber) is called local time base. A system of physical clocks that are synchronized with themselves and are located at different locations (i.e. at different subscribers) in order to determine a consistent time assessment is called global time base. To adjust the local clocks of the global time base a number of microticks are merged to a macrotick because it is not possible to synchronize clocks perfectly[Kop97]. All subscribers use the same global time base and every subscriber has its own local time base. Time bases use counters in order to take snapshots of time points and each such snapshot represents a time stamp. Time stamps of local time bases cannot be compared to each other because they are not synchronized. This is not the case for time stamps that are taken at any locations from the global time base and time stamps of the same local time base.

Messages and Triggering Messages

A message is named by the identifier of the message and the priority of a message is indicated by the identifier such that a lower number is higher in priority. Messages are periodically created in time intervals and each time interval in between two messages with the same identifier is called Message Time Interval (MTI). Each MTI consists of a static period that is called Minimum Interarrival Time (MINT) and a variable random time from 0 up to a predefined upper bound called Random Send Time (RAND). The MINT and RAND are defined by a number of time quanta and a time quantum is measured by the local time base. Therefore each MTI is coupled to an identifier and named by its identifier. At the creation of a message i the MTI i is recalculated.

A Message Creation Entity (MCE) that consists of an identifier is a unique number from 1 up to the total number of MCEs of all subscribers and a MCE is named by the identifier of the MCE. The MCE also includes a MINT, a RAND and a sequence number. In addition, a MCE is always related to a MTI that is calculated by the MINT and RAND of the corresponding MCE. Moreover, each subscriber has a bundle of MCEs. Furthermore, a Triggered Message Instance (TMI) is a message that is derived from a MCE such that the identifier of the message is the identifier of the corresponding MCE. The payload of the message consists of a time stamp from the global time base that is obtained when the message is created. The time stamp that is used as payload for a TMI is called Trigger Time of a Message (TTM) and consists of 8 bytes in order to assess the accurate time point. The creation of TMIs are triggered by the local time base in such a way that the local time base indicates the end of a time quantum and at the end of each time quantum all MTIs are checked to be finished. A TMI is named by its identifier. A TMI i In addition, with the next MTI i that triggers the next TMI i are created at the end of the actual MTI i that is calculated by the RAND and MINT of the MCE i. At the creation of a TMI i the sequence number of the MCE i is incremented. Figure 4.2 shows the periodic creation behaviour of a MCE i after n-1 time intervals.

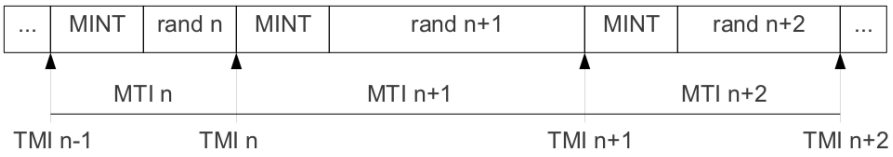


Figure 4.2: Message Time Intervals of a Message (TMI)

The specification of an application defines the minimum time between messages of the same identifier and the MINT stands for the minimum time. External influences and/or the state of the system stochastically extend the minimum time between messages of the same identifier and the RAND represent the random time. Thus the MTI simulates the time that is required by a specific process to create a message that is ready for transmission. By assembling some MCEs with their associated MINTs and RANDs the communication behaviour of a CAN node is simulated (see Subsection 3.1). A TMI is then triggered by a local time base due to a node and therefore its processes are independent from other locations and other clocks.

COM Interface and Communication Errors

Each TMI is transported via a data frame. All data frames are transported via a CAN Controller (COM Interface). In CAN the COM Interface is switched off if an error counter for sending of the COM Interface exceeds a allowed limit (see Subsection 3.1). In such a case this will lead to an inconsistent reception of TMIs at the subscribers as well as to the absence of emitted TMIs because each TMI is only sent/received if the COM Interface is active. Furthermore, the COM Interface has reduced communication privileges if the error counter for sending or the error counter for receiving exceeds a defined limit and the COM interface is not switched off. Thus these error counters have to be checked periodically.

An inconsistent reception of TMIs also occurs by bit flipping faults at the arbitration field such that an unused identifier will be emitted or an identifier of another TMI will be simulated. Such simulated faults have to be detected in a post data evaluation process by comparison of incoming TMIs at the subscribers to the sequence numbers as well as by the identifiers of the corresponding MCEs. Bit flipping faults also effect the atomic broad cast of a message if the last bit of a CAN frame is read differently by the receivers. In such an instance the message will be accepted by some receivers and rejected by others[PV03] (see Subsection 3.1). In addition, a CAN star may be erroneous and therefore not forwarding incoming traffic (beside routing) to all corresponding ports. This types of error will also be detected by a post data evaluation process of the sequence numbers from the MCEs.

Transmission and Transmission Times of Messages

A created TMI is placed into a Prioritized Send Queue (PSQ) that is a queue with entries for each MCE. A PSQ is ordered in two ways; firstly according to the priority of the TMIs in the PSQ (high priority before low priority) and secondly to the temporal order of the TTMs of the stored

TMIs such that TMIs with the same identifier are sorted by their time stamps in ascending order. The PSQ supports for every MCE entries up to a predefined limit and if the PSQ is completely filled for a MCE each new TMI of that MCE is discarded and counted by a sequence number for that MCE that is called MCE Omission Number (MON).

A PSQ that is located at the node and at the COM interface of a subscriber features a Single Message Spot (SMS). Importantly, a SMS can be used to store a single TMI from the PSQ for transmission. A free SMS is filled with the first entry of the associated nonempty PSQ and so the TMI with the highest priority is moved from the node to the COM Interface in order to transmit it to other subscribers as fast as possible.

For example, assume a number of TMIs (a TMI i created at time point ts is denoted as TMI i,ts) placed in a PSQ with a limitation of 3 TMIs for each MCE in the following order: TMI 1,12; TMI 1,15; TMI 3,3; TMI 3,12; TMI 3,20. When a single TMI is moved to the SMS and the TMI 2;30 and TMI 3;30 will be generated the PSQ changes as follows: TMI 1,15; TMI 2,30; TMI 3,3; TMI 3,12; TMI 3,20.

A transmission in CAN may start if the (sub)network is idle and succeed if it achieves the arbitration process (see Subsection 3.1). In this case the transmission is not rejected by any receiver or sender. The COM Interfaces are configured in such way that they accept all identifiers and so any TMI (see Subsection 3.1). Therefore a TMI that is emitted by a SMS in a network is received by all inactive COM Interfaces and a COM Interface supports a First In - First Out (FIFO) queue to store received TMIs that is called Receive Message Queue (RMQ). RMQs are periodically checked and if an incoming TMI is recognized it is moved to the node and a time stamp from the global time base is taken. Such a global time stamp is called Message Reception Time (MRT). At a subscriber the corresponding MRT is attached to the received TMI. A TMI together with its coupled MRT is called Received Message Instance (RMI). A RMI is named by its identifier. All subscribers act as receivers and so a TMI is converted to multiple RMIs that consist of an identifier, a TTM and a MRT. However CAN star subscribers that are not connected to the transmission medium of the sender but to a port of the CAN star may not receive the message because the message is not forwarded to the port or dropped if the CAN star is overloaded (see Subsection 3.2).

SMS processing includes the time for conversion of a message into a frame, the block time of a TMI by a busy network and retransmissions due to errors. The reception time of a message depends on two main issues; firstly, the conversion time from frame to message and secondly, the access time to the COM Interface by the subscriber in order to catch the message. The difference between MRT and TTM is called Total Transmission Time (TTT) of a RMI and represents the time for waiting in a PSQ, processing in the SMS, transmission from a sender to the receivers and reception at a receiver. Furthermore, the transport of a TMI in a CAN star also consists of the routing time and the possible retransmission at the ports to the subnetworks. This corresponds therefore to an end-to-end transmission time. Figure 4.3 shows the transmission process of a message from a sender to a single receiver. In a CAN bus the TTTs of RMIs from a TMI regardless of a short time between the reception and recognition of a message are equal. This can be assumed because each message is received simultaneously. Nevertheless, in CAN

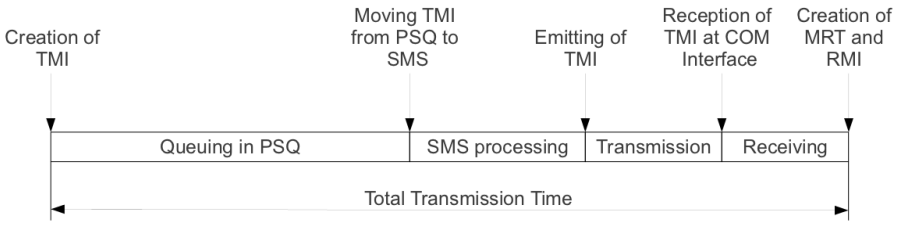


Figure 4.3: Transmission and Transmission Time of a Message

star the TTTs vary due to the repetition of message emission at the ports of the CAN star.

TMIs in the PSQ are prioritized by their identifiers such that a lower identifier has a higher preference to provide an accurate simulation of a CAN system. This is because CAN uses CSMA/CR that also priorities the lowest identifier of collided CAN frames at the transmission medium.

The properties of the PSQ also imply that it is possible that TMIs of all MCEs can be presented in the system. If not, high priority TMIs can be blocked by low priority TMIs at a subscriber because low priority TMIs can completely fill up the PSQ. Therefor the spots in the PSQ are limiting each MCE to avoid starvation such that TMIs from all MCEs can be present in the PSQs.

CAN Simulation Device

A subscriber that is attached to a communication medium (CAN star or CAN bus) and used as a device to simulate a CAN subscriber is called CAN Simulation device (CSD). Figure 4.4 shows a block diagram of a CSD. The Local Time Base unit triggers the TMI Trigger unit to generate TMIs (see Subsection 4.1) that are moved to the Prioritized Send Queue unit. This Prioritized Send Queue unit is used for queuing as explained in Section 4.1. The TMI to RMI Converter unit catches incoming TMIs to convert them to RMIs as mentioned in Subsection 4.1. Such objects are stored in the RMI Storage unit for further statistical evaluations. The COM Interface is part of a CSD and the Error Check Unit observes the internal state of the COM Interface to ensure the correct communication behaviour such that the CSD is an active member in the CAN communication system (see Subsection 3.1) and no incoming TMIs are lost. The TX unit converts TMIs from the SMS unit to frames and emits them to the communication medium. In addition, the RX unit receives frames from the communication medium and converts them to TMIs as exemplified in Section 4.1. The Receive Message Queue (RMQ) unit temporally stores incoming TMIs from the RX unit until they are removed by the TMI to RMI Converter unit as clearly described in Section 4.1. The Error Signalling unit detects errors upon transmission. Furthermore, it also signals them to the transmission medium by error frames (see Subsection 3.1), turns off the communication (reduces communication privileges) as required by the CAN standard, checks the RMQ for overruns and provides the internal state of the Communication Interface to the Error Detection unit. The SMS unit stores a message to be transmitted via the

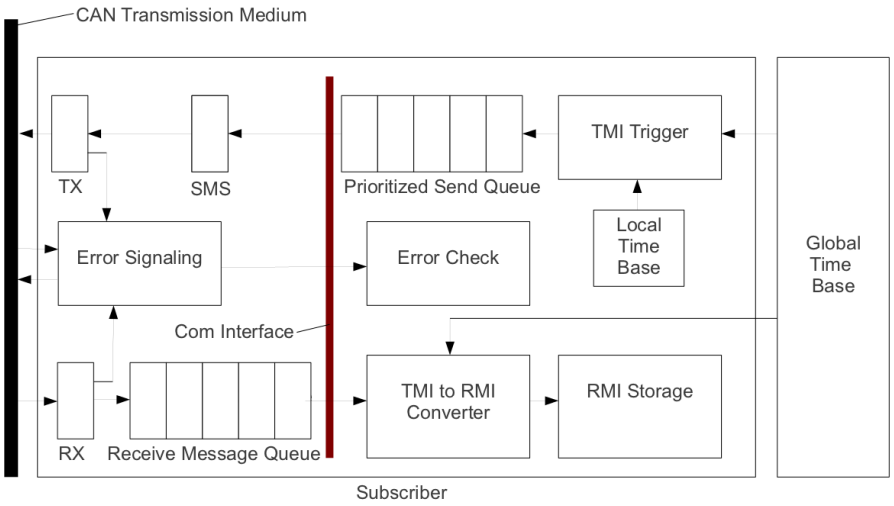


Figure 4.4: Block Diagram of a CAN Simulation Device

communication medium.

The COM Interface provides communication services that are used in the CSD. The CSD makes use of cycles to simulate a CAN device and each cycle is composed as follows:

- Error check phase to scan the Error Signalling unit for an erroneous state that is if the COM Interface is switched off (has reduced communication privileges) due to errors or an overflow at the RMQ at the COM Interface. If such an erroneous state is detected at the Error Check unit the CSD has to stop its services and so the error check phase assures the sound condition of a CSD as well as the not absent of incoming TMI in case of an overrun at the RMQ.
- Listen phase to check the RMQ for incoming TMIs to empty the RMQ. TMIs from the RMQ are converted to RMIs and placed in the RMI Storage by the TMI to RMI Converter.
- Sending phase to check the PSQ for TMIs. If the SMS is free and the PSQ is occupied the TMI that was first in order in the PSQ will be moved to the SMS.

Each cycle starts with the error check phase, followed by the listen phase and continues with the sending phase followed by the next cycle. Each cycle follows the same procedure. A cycle is periodically interrupted by the local time base to determine TMIs that are placed to the PSQ as described in Section 4.1. Therefore the generation of TMIs is independent from the emission and reception behaviour of a CSD.

Test System

A test system consists of an arbitrary number of CSDs, a communication medium, a global time base and a central unit. The central unit has a dual function, it evaluates statistical data

and controls the behaviour of the CSDs and is therefore called Central Evaluation Unit (CEU). The CSDs are attached to the communication medium and the global time base and the CEU is connected to the CSDs. The test system simulates a CAN network by test rounds. The test round is a process with a defined start and end and simulates a CAN network over time. The CEU activates the CSDs at the start and monitors the sound condition of the system. Hence, the CEU assures that each CSD ends a test round in time, that it starts simultaneously with other CSDs, that it does not interrupt its services and that it is functioning properly. A test run consists of a number of test rounds in order to evaluate a CAN network under different utilization or to improve the statistical explanatory power of a test configuration. If the test system was impaired during a test round a rerun has to be performed.

Statistical Data

The TTTs are fluctuating due to the internal states of the CSDs as well as the utilization of the transmission medium. If there are queues in the PSQ the internal state of a CSD will be affected in such a way that queuing results in jitter (the jitter is defined as the difference of the total maximum transmission time and minimum transmission time) for the transmission times of TMIs. The PSQ queues also interfere with the utilization of the communication medium and other TMIs that were queued in other PSQs. If a TMI is emitted the communication medium for all other TMIs that have to be sent off will be blocked and thus the TTTs for these TMIs will increase. Furthermore, these TMIs that are waiting for transmissions reoccupy the communication medium. During the occupation of the communication medium TMIs can be triggered that elongate the block time of the communication medium and so the TTTs will increase. In contrast in a system without queued TMIs any single generated TMI will be delivered without delay. Therefore the jitter can be dramatically increased even if the average system utilization is low. In an Event-Triggered system these cases occur frequently because the communication behaviour depends on statistical properties.

In addition, the TTTs of RMIs from each TMI can vary as the COM Interfaces are accessed independently. Each TMI can be received differently in time at a particular CSD. Moreover, the TTTs of RMIs of a TMI sent over a CAN star also vary due to reemissions. The fragmentation of networks into subnetworks may also impact the internal states of all CSDs due to varying utilization of the corresponding subnetworks. Therefore the following statistical data are evaluated for each CSD in order to determine the timely behaviour during a test round as follows:

- Minimum transmission time for each MCE to determine the fastest transmission
- Maximum transmission time for each MCE to determine the slowest transmission
- Average transmission time for each MCE to determine the average transmission time
- Variance of the transmission time for each MCE to determine the distribution of transmission times for each message id

As mentioned in Section 4.1 TMIs may not be transmitted to all CSDs and so the number of sent and received TMIs has to be counted to determine inconsistent broad casts and to determine

the average utilization of the communication medium. The number of TMIs that are created but not squeezed into a PSQ or that have not been sent off must be considered in order to detect system overloads. Therefore the following statistical data are evaluated:

- The number of sent TMIs for each MCE
- The number of received TMIs at each CSD from each MCE
- The number of TMIs for each MCE that were created but not sent off (called send omissions). Send omissions are defined as the sum of MONs and the number of TMIs that were located in a PSQ at the end of a test round.
- The difference between the emitted TMIs and the received TMIs at each CSD (receive omissions).

4.2 Experimental Model

In a test run MCEs are distributed over all CSDs and the behaviour of the test system is mainly influenced by the ids of the MCEs. The reason for this is because the ids of the MCEs reflect the priority of messages that are sent in a CAN network and send frequencies that are determined by MINTs and RANDs of the MCEs. If nothing else is specified all times are indicated in μs.

Basic Definitions

There are n MCEs from 1 to n that are distributed over CSDs in such a way that a particular CSD (Ramp CSD) owns a single MCE with the lowest or highest priority that is called Ramp MCE. The remaining MCEs are evenly distributed over the remaining CSDs and therefore numbered from 0 to m. In conclusion, in a test system m+1 CSDs exist and a CSD i has all MCEs with identifiers (n-1)modulo i but not the Ramp MCE.

The MINT and RAND of a MCE i are marked by $MINT_i$ and the $RAND_i$. A Start Minimum Interarrival Time (SMINT) that defines the MINT for the lowest priority MCE and a Bandwidth Step (BSTEP) (difference between MINTs of MCE i and MCE i+1) generate the $MINT_i$ of a MCE i for the first test round in a test run as follows:

$$MINT_i = SMINT + (i-1) * BSTEP \tag{4.1}$$

In the following test rounds the MINTs of the MCEs are changed if the MINT is from the Ramp MCE that is called Ramp Minimum Interarrival Time (RMINT). In this case the RMINT of a test round j is denoted as $RMINT_j$. The first test round in a test run obtains number 0. The difference between $RMINT_j$ and $RMINT_{j+1}$ is called Round Bandwidth Step (RSTEP) and a $RMINT_j$ is defined for a test round j by the RSTEP as follows:

$$RMINT_j = RMINT_0 - j * RSTEP \tag{4.2}$$

A $RAND_i$ is defined by $MINT_i$ as follows:

$$RAND_i = 2 * MINT_i \tag{4.3}$$

The average trigger time of a MCE i is called $AVERAGE_i$ and corresponds to $RAND_i$. On average a TMI is triggered after the MINT that is half a RAND in addition to a stochastic time quantum that is also half a RAND:

$$AVERAGE_i = MINT_i + \frac{RAND_i}{2} = RAND_i \tag{4.4}$$

To calculate the bandwidth consumption of a message the length of a data frame at the transmission medium has to be estimated. In the test setup a payload of 8 byte is used. In addition of the overhead in a CAN frame and the intermission time of 3 Bits that is required by the CAN standard, a data frame consumes 131 Bits at OSI layer 2. Bit stuffing at OSI layer 1 can be estimated to be 4 Bits and so the total CAN data frame requires 135 Bits at the transmission medium. Therefore the bandwidth consumption in $kbit/s$ for a TMI i ($bandwidth_i$) is defined as follows:

$$bandwidth_i = \frac{\frac{1}{Average_i} * 135}{1024} \tag{4.5}$$

The bandwidth consumption of the entire CAN network is called
Total Bandwidth Consumption (TBAND) and is the sum of bandwidth consumption from all TMIs:

$$TBAND = \sum_{i=1}^{n} bandwidth_i \tag{4.6}$$

Test Setups

To evaluate the properties of the CAN standard different configurations are chosen to compare the behaviour of CAN networks under different conditions. The test setups consist of a CAN bus that can be accessed by 4 or 8 CSDs at a network speed of 500 kbit/s. The granularity of the local time base is 100 μs and of the global time base 0.0745 μs. The PSQ has a length of 32 entries for each MCE and a CSD i obtains 10 MCEs. Therefore in a test setup with 4 CSDs there will be 31 MCEs and in analogy 71 MCES with 8 CSDs. Furthermore, each test round has a duration of 10 s.

The basic utilization of each test run is 40 percent (determined by TBAND) and the test systems are evaluated with a RSTEP of zero and a RSTEP of nonzero. A test run with a RSTEP of zero evaluates a CAN network with a typical utilization of 40%[DBBL]. A number of test rounds that are identically configured assures a significantly increased explanatory power. The test run with a RSTEP of zero and 4 CSDs is called '4 CSDs - No Ramp MCE' and the test run

Number of CSD	Ramp MCE	SMIN	BSTEP	RSTEP	Test Rounds
4	0x1	8000	150	0	800
8	0x1	8000	600	0	800
4	0x1	8000	150	10	800
4	0x1f	8000	150	10	1250
8	0x1	8000	600	1	8000
8	0x47	8000	600	50	1000

Table 4.1: Test run configurations

with a RSTEP of zero and 8 CSDs is called '8 CSDs - No Ramp MCE'. The variation in the number of CSDs shows the impact of a different number of CAN devices on a communication medium and a different number of messages (ids). Furthermore, a test run with a RSTEP of nonzero simulates a CAN network under different utilizations. The test runs with RSTEP of zero can then be compared with the test runs having a RSTEP of nonzero. Moreover, the test run with the Ramp MCE with the highest priority and 4 CSDs is called '4 CSDs - High Priority Ramp MCE' and the test run with the Ramp MCE with the highest priority and 8 CSDs is called '8 CSDs - High Priority Ramp MCE'. In addition, the test run with the Ramp MCE with the lowest priority and 4 CSDs is called '4 CSDs - Low Priority Ramp MCE' and the test run with the Ramp MCE with the lowest priority and 8 CSDs is called '8 CSDs - Low Priority Ramp MCE'. The variation in the number of CSDs shows the impact of a different number of CAN devices on a communication medium and a different number of messages (ids). Table 4.1 shows all test runs and their specific configuration.

In all test runs the Ramp CSD only has a single MCE due to the architecture of the test system. The Ramp CSD also eliminates the interconnections of the MCEs from a single CSD. The Ramp MCE has the highest or lowest priority as CAN is a priority based system. The increase in bandwidth of the Ramp MCE is correlated with to an elevated utilization that shows the consequences of applying a CSMA/CR technique. A CAN star is not tested as this work tries to focus on the CAN bus.

CHAPTER 5

Prototype Setup and Experiments

This chapter describes the structure of the prototype and the overall test system. In addition, the design toolchain and the test system toolchain will be explained. Furthermore, the structure of the timing model and experiments (test runs) will be outlined. Moreover, this chapter explains the test data preparation that is used in Chapter 6.

5.1 Explanation of Structure

The evaluation of the CAN protocol is performed by a HIL-method (see Subsection 3.3). Therefore the CAN bus including the CAN Transceivers (real hardware) interacts with the CSDs (controller model) over time (see Figure 3.9). Note that the CAN Controllers are part of the CAN bus/protocol and that they can be seen as part of the real hardware. Furthermore, the CEU controls this process and evaluates the data (received from CSD) in order to generate statistical data of the transmission time behaviour of CAN messages (see Subsection 4.1). The CSDs as well as the CEU form a system that evaluates the CAN protocol. This system is placed onto a FPGA (Altera Stratix III[Alt08]) and therefore this system is a SoC (see Subsection 3.4). Furthermore, the FPGA is part of the FPGA-prototype board that is a construction base for developing FPGA projects. Each FPGA-prototype board provides additional hardware to the FPGA e.g. I/O. In addition, each FPGA-prototype board provides a programming interface for the developer via a host computer. Thus a SoC that is placed onto the FPGA-prototype board to evaluate the transmission time behaviour of the CAN protocol forms a prototype. Figure 5.1 shows the architecture of the prototype and the overall test system. The prototype board communicates via a High-Speed Mezzanine Cards (HSMC)-interface (an interface that supports signalling to external devices[Alt08]) with the attached CAN Transceivers (Tr). The CAN Transceivers are interconnected by two unshielded copper wires (bus) which are connected by two 120 Ω resistors (see Subsection 3.1). Furthermore, the FPGA also communicates with a host computer via a serial interface (an interface for a serial communication) in order to 1) setup the prototype, to 2) setup the configuration of each test round and to 3) store statistical data created by a run of a test round.

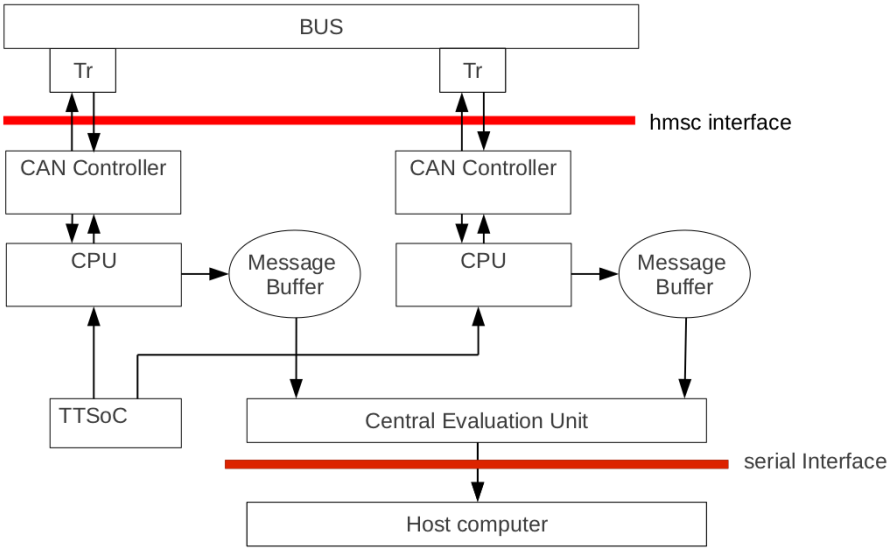


Figure 5.1: Interfaces of the prototype

The SoC consists of IP cores (see Subsection 3.4) that implement the system features. The system features are divided into blocks as follows:

- CAN Controllers (see Subsection 3.1) which are realized by IFI Avalon CAN Modules[Ing09]. Each CAN Controller acts as a COM Interface.
- Message Buffers and each Message Buffer is realized as a self-developed IP core (written in VHDL). Each Message Buffer also acts as a RMI Storage (see Subsection 4.1).
- TTSoC (see Subsection 3.4) that is placed on the FPGA. The TTSoC provides the global time base (see Subsection 4.1) via its TISSes to the CPUs.
- CPUs which are realized by a Nios II/e CPU IP cores[Alt09]. Each CPU together with its associated CAN Controller, Message Buffer, TISS and interrupt timer (local time base, not shown in Figure 5.1, see Subsection 4.1) form a CSD (see Subsection 4.1). Each CSD connects to a CAN Transceiver to interconnect the CSD with each other. Furthermore, there are more IP cores connected to the CPUs (see Subsection 5.2).
- CEU (see Subsection 4.1) that is realized by a Nios II/f CPU IP core[Alt09]. The CEU receives RMIs from the message buffers to calculate the statistical data of each test round. Furthermore, the CEU controls each test round (not shown in Figure 5.1).
- Synch-Unit (not shown in Figure 5.1) that is memory to synchronize the CSDs by the CEU. Each CSD does not act as an active communication member until the CEU writes a code word into the Synch-Unit. The Synch-Unit is continuously read before the start

of each test round from all CSDs and a particular CSD starts to act as an active communication member if the codeword was read correctly. Thus from the point of view of a particular CSD the test round starts by reading the correct codeword.

The IP cores are interconnected via the Avalon Interconnect (see Subsection 3.4). Furthermore, the CPUs (including the CEU) with their attached hardware (IP cores) form a Multiprocessor System-on-Chip. Furthermore, the prototype can be realized by an arbitrary number of CSDs because the FPGA is reconfigurable (see Subsection 3.4). However the test runs always use the same prototype configuration consisting of 8 CSDs. In test runs consisting of 4 CSDs the remaining CSDs are switched off. Furthermore, the system frequency is set to 100 MHz.

5.2 Explanation of Structure Elements

This section describes the particular system components and the host computer in detail.

CAN Transceiver

A MCP2551-I/P[Mic03] is taken as a CAN Transceiver. The TX pin of the MCP2551-I/P connects via a line of the HSMC-interface to the TX connector of the CAN Controller. The RX pin of the MCP2551-I/P connects via a line of the HSMC-interface to the RX connector of the CAN Controller. Note that this is not a direct wiring because there are two amplifier circuits between the MCP2551-I/P and the connection to the HSMC-interface in order to obtain a proper voltage level at the CAN Transceiver (5 V) and the HSMC-interface (2.5 V).

CAN Simulation Device

This Subsection describes the structure of the CSD for the prototype implementation. Each CSD consists of a processor with attached IP cores.

Processor

The Nios II/e IP-Core processor is taken as a controller to perform the logical operations to simulate the behaviour of a subscriber e.g. message generation. Furthermore, this processor configuration was taken due the low consumption of logic. Each FPGA provides a limited number of logic. Furthermore, the processor was extended by the IP cores beside the Message Buffer, TISS, Interrupt Timer and CAN Controller as follows:

- JTAG UART that is communication interface to program the processor and for the information exchange. Furthermore, the JTAG UART interacts with the serial interface such that the JTAG UART is connected to a host computer (see Subsection 5.2). The JTAG UART is used to program (the MCEs are configured during programming) the CSD.
- PLL to receive a proper system frequency.
- On-Chip memory to store the program of the processor and execution data.

- Synch-Unit (memory) to determine the start of each test round.

Message Buffer

The message buffer has two Avalon Memory Mapped Interfaces (see Subsection 3.4). The first interface acts as a RMI storage of the CSD (access the interface to store RMIs) and the second interface connects to the CEU in order to obtain the RMIs which were stored by the CEU in the Message Buffer. The Message Buffer is realized as a ring buffer. Therefore the stored data are placed in a queue and when the queue ends the following data is stored in first place of the data queue. In addition, the CEU reads the data from the queue after the data was placed. Thus data can be only overwritten if the corresponding data was loaded. Furthermore, the ring buffer also acts for transmission of additional information about each MCE and for transferring control information e.g. send number of each TMI.

TISS

A TTSoC (see Subsection 3.4) was placed onto the FPGA and each TISS of the TTSOC is attached to the processor of each CSD. Each TISS is used to achieve time stamps from the global time base for measuring time. At each creation of a message (TMI) a time stamp is taken from the TISS of the sender. Upon at the reception of the message at a particular CSD (not the sending CSD) a time stamp from the TISS of the corresponding receiver is taken. Hence the transmission time can be calculated from the difference between these two time stamps.

Interrupt Timer

The interrupt timer is used to obtain the local time base.

CAN Controller

An IFI Avalon CAN Module was taken to serve as a CAN Controller. The CAN Controller is the COM Interface (see Subsection 4.1) of the CSD and therefore it provides the SMS, RMQ and Error Signalling (see Subsection 4.1). The SMS is realized by the FIFO transmission buffer of the CAN Controller such that the FIFO transmission queue has a single entry. Moreover, the RMQ is realized by the receive message queue of the CAN Controller. Furthermore, the CAN Controller manages the CAN traffic as explained in Subsection 3.1. Moreover, the CAN Controller accepts all received message. Therefore all received TMIs are accepted (see Subsection 3.1).

Behaviour of the CSD

The processor runs a main program (written in the C-programming language) and has additional functions that enable it to act with its attached IP cores as CSD. Furthermore, the main program starts with the configuration of the CAN Controller (COM Interface) with the desired speed. In addition, the interrupt timer gets configured such that the local time base is adjusted. Moreover, the first MTI of each CSD is calculated. The program continues with continuous reading of the

Synch-Unit and holds on until a desired code word is read. Furthermore, the CSD informs the CEU via its message buffer about its start time (global time base) and starts the local time base. At this time point the main loop starts.

The main loop runs as described in the Listing 5.1

Listing 5.1: Main loop of the CSD

```
start=get_time(); /*get star time of test round
                    (global time base)*/
while(get_time()-start <= TEST_ROUND_TIME)//test round duration
        for(i = 0; i < MCE_ON_CSD; i++) //for each MCE i
                if(CAN_error())
                        error_handling(CAN_ERROR);/*
                        Error check unit*/
                else if(TMI_arrived()){//TMI arrived
                        receive();      /*
                        receive TMI and create RMI*/
                        break;/*start with highest priority
                        MCE*/
                }
                else if(SMS_full())//Check SMS to be free
                        break;//start with highest priority MCE
                /*check PSQ for TMIs of the actual MCE*/
                else (MCE[i].reader != MCE[i].writer){
                        /*push TMI into SMS*/
                        send_TMI(i,MCE[i].time_stamp\
                        [MCE[i].reader]);
                        MCE[i].reader++;
                        MCE[i].reader%=QUEUE_LENGTH;
                        /*increment send number of
                        the actual MCE*/
                        MCE[i].send_number++
                        break;//start with highest priority MCE
                }
```

The main loop starts at the beginning of the test round and finishes at the end of the test round. MCEs are checked for triggered TMIs according to their priorities. Furthermore, each loop run (MCE check) starts with a control of the CAN Interface to guarantee that the CAN Controller is in the Error Active state (see Subsection 3.1). If the CAN Controller is not in the Error Active state, the test round is aborted and the CEU is informed by the error_handling-function. Therefore the error_handling-function realizes the Error Check unit of the CSD.

After a successful error check phase the RMQ is checked for received TMIs and if a TMI is received, the receive-function (see Listing 5.3) transforms the TMI into a RMI, places it into its message buffer (RMI Storage) and the main loop continues at the start.

If the RMQ was empty, the SMS will be checked for vacancy. If the SMS is not free, the loop continues at the start.

In case that the SMS was empty the PSQ corresponding to the actual MCE is checked for triggered TMIs. The PSQ it self is realized as numbers of ring buffers. Each MCE has its own ring buffer and during the run of the loop coordinated by the priorities of the MCEs, the PSQ is realized as explained in Subsection 4.1. Furthermore, the ring buffer itself consists of a number of entries for time stamps from the global time base. At the generation of a TMI (see Listing 5.2) the current time stamp is moved into the ring buffer and the corresponding write counter will be incremented or reset to the first position. If a time stamp is taken from the ring buffer, the corresponding read counter will be incremented or reset to the first position. Furthermore, the loop count (MCE number) implies the id of the TMI. If the PSQ is not empty for the corresponding MCE (the MCE with lower priority was checked before) the first entry will be taken. This process is determined by the read counter position. This entry will therefore be moved in the SMS with the id of the TMI. Furthermore, the send number of the corresponding MCE is incremented and the loop continues at the start.

After the end of the main loop the COM Interface is checked for TMIs which arrives after the round end. These TMIs are also transformed into RMIs and placed into the Message Buffer (RMI Storage) by the receive-function. Moreover, the send omissions and send numbers of the MCEs of the particular CSD are submitted to the CEU. After this information transfer the CEU is informed of the completion of the corresponding CSD. These informations are placed into the message buffer.

The behaviour of the TMI trigger unit is shown in Listing 5.2. The TMI trigger unit is realized as an interrupt service routine and raises as defined by the time quantum of the local time base (interrupt interval). Furthermore, the MTI (see Subsection 4.1) is checked for each MCE. If a particular MTI advances to zero, the PSQ is checked for an empty entry of the corresponding MCE. If there is an empty entry, the TMI is moved into the PSQ. As mentioned above the PSQ is realized by a number of ordered ring buffers. Each TMI is moved into the PSQ and therefore in the corresponding ring buffer according to the storage of the actual time stamp of the global time base at the first free entry. If it is free, the time stamp is copied into the ring buffer and the corresponding write counter is incremented by 1 or reset to 0. The id of the TMI is implied by the MCE (data structure). Furthermore, if the PSQ is full, the corresponding MON is incremented and the latest time stamp is not taken into account. Furthermore, if the MTI is not zero, it gets advanced in time by the time quantum of the local time base (subtraction).

Listing 5.2: IRQ of the CSD

```
for(i = 0; i < MCE_ON_CSD; i++) //for each MCE i
        /*check MTI of each MCE*/
        if(MCE[i].send_time <=0){//MTI advanced to zero
                /*calculate next MTI*/
                MCE[i].send_time = rand()%MCE[i].RAND;
                MCE[i].send_time += MCE[i].MINT;
                MCE[i].send_time -= TIME_QUANTUM;//advance time
                /*push TMI into PSQ*/
                if(MCE[i].writer+1)%QUEUE_LENGTH != \
                        MCE[i].reader){
                        MCE[i].time_stamp[MCE[i].writer]=\
                        get_time();
                        MCE[i].writer++;
                        MCE[i].writer%=QUEUE_LENGTH;
                }
                else
                        /*PSQ full for that MCE*/
                        MCE[i].send_omissions++;
        }
        else
                MCE[i].send_time -= 100;//advance MTI in time
```

The behaviour of the TMI to RMI Converter unit (receive-function) is shown in Listing 5.3.

Listing 5.3: Receive function of the CSD

```
void receive(){
        /*get receive time(global time base)*/
        receive_time = get_time();
        receive_TMI(&i,send_time);//get TMI from RMQ

        /*message buffer check such that there is no delay*/
        if(free_message_buffer)
                /*create RMI and push it into RMI storage*/
                write_message_buffer(C0,receive_time, \
                send_time,i);
        else
                /*inform CEU about error*/
                error_handling(RMI_STORAGE_ERROR);
}
```

After the detection of a TMI in the RMQ the receive-function is called. The receive-function takes the first TMI from the RMQ and attaches the actual time stamp of the global time base. Therefore a RMI is created (see Subsection 4.1). Next, the message buffer is checked for accessibility. If this event does not occur, the error-handling-function will inform the CEU about

an error. This technique guarantees that the RMI Storage unit (CEU) does not affect the timing behaviour of the CSD e.g. send and receive TMIs. If the check up of the accessibility of the message buffer was successful, the corresponding RMI will be pushed into the message buffer (RMI storage).

Central Evaluation Unit

This Subsection describes the structure of the CEU in the prototype implementation. The CEU consists of a processor with attached IP cores.

Processor

The Nios II/f IP-Core processor is taken as a controller to perform logical operations in order to 1) receive RMIs from the CSDs, to 2) control the overall MPSoC system (see Subsection 3.4), to 3) create the statistical data of the transmission time behaviour (see Subsection 4.1) and 4) to send the statistical data to the host computer. Furthermore, this processor configuration was taken because the Nios II/f is the fastest processor of the Nios II-Family. The CEU has to receive RMIs from up to 8 CSDs and a message buffer overrun denotes an erroneous system. Thus the CEU has to react fast and hence a fast CPU-type is required. In addition, the CEU evaluates the statistical data and a fast CPU reduces the time for such computations and therefore the total test run time. Furthermore, the processor was extended by the IP cores as follows:

- JTAG UART that is used to program the CEU with additional parameters. In addition, the JTAG UART is used to transmit the calculated statistical data to the host computer.
- PLL to receive a proper system frequency.
- On-Chip memory to store the program of the processor and execution data.
- Synch-Unit to start the CSDs in parallel (synchronization of CSDs).
- Message Buffer to receive the RMIs from the CSDs, statistical send data of each MCE and control information from each subscriber e.g. start time of the test round. Each CSD connects to a separate message buffer that is connected to the CEU.
- DDR2-Memory (memory-controller) to store the received RMIs during the test round.
- Interrupt Timer to assure that each CSD completes (not stuck) in time.

Behaviour of the CEU

The processor runs a main program (written in the C-program language) and additional functions in order to act with its attached IP cores as CEU. Furthermore, the main program starts with the configuration of the interrupt timer. Moreover, the CEU activates all CSDs by writing a code word into the Synch-Unit-unit and continues with the activation of the interrupt timer. At this time point the main loop starts.

The main loop runs as described in the Listing 5.4. The main loop runs until each CSD sends its finish signal. Furthermore, the message buffer is read by the read_message_buffer-function in order to receive RMIs, statistical data and control signals. A specific signal is determined by a code word, that is stored with the corresponding data in the message buffer e.g. C0 means, that the corresponding data forms a RMI. Furthermore, the write_DDR2-function stores a RMI in the DDR2-memory. In addition, the store_MCE_stat-function stores send omissions and send number of each MCE. Moreover, the start_time-array is used to store the start time of each CSD. Furthermore, the error_handling-function performs the exception handling to inform the host computer about errors e.g. a CSD is not in the Error Active state.

Listing 5.4: Main loop of the CSD

```
run = 1;//run parameter of main loop
end_count = 0;//how many CSD have completed

while(run)
        for(i=0; i < CSD_NUMBER; i++)//check each CSD
                if(msg_buffer[i].filled()){//Data available
                        /*read message buffer*/
                        read_message_buffer(w1, w2, w3, w4);

                        if(w1==C0)//RMI
                                write_DDR2(w2, w3, w4);
                        else if (w1 == C1)/*
                                        send omissions, etc.*/
                                store_MCE_stat(w2, i);
                        /*Start time from each CSD*/
                        else if (w1 == C2)
                                start_time[i] = w2;
                        /*FIN signal from each CSD*/
                        else if (w1 == C3){
                                end_count++;
                                if(end_count == CSD_NUMBER)
                                        run = 0;
                        }
                        /*CSD informs CEU about an error*/
                        else if (W1 == C4)
                                error_handling(w2);
                        else
                                /*something unexpected*/
                                error_handling\
                                (UNEXP_MSG_FORMAT);

                }
```

After the end of the main loop, the interrupt timer is turned off. If the main loop does not interrupt, the interrupt timer does not get switched off. Therefore an IRQ raises such that the host computer gets informed about the erroneous state (stuck CSD). Furthermore, the Synch-Unit gets reset for the next test round. Next, the RMIs are checked for their identifiers such that each identifier is a MCE id. In addition, all RMIs that were sent after the test round end are removed. The test round end is the sum of the latest start time of the start_time-array (see Listing 5.4) and the test round time. Furthermore, the statistical data are calculated as defined in Subsection 4.1 and transmitted to the host computer. In addition, the CEU sends the start and end times of the test round and the removed RMIs to the host computer.

Host Computer

The host computer in the actual implementation is a personal computer and uses a Unix operating system (Linux) to manage the whole design process as well as the experiments. Furthermore, each test run is setup and managed by a host computer. Therefore the number of test rounds, definition of the RAMP MCE, number and definition of MCEs, number of CSDs, SMINT, BSTEP and RSTEP (configuration of all MINTs and RANDs for each test round - see Subsection 4.2), definition of the granularity of the local time base and global time base, setup of the CAN Controller (COM Interface) and test round time are defined via the host computer. Furthermore, the host computer controls each test round. Therefore the host computer sets up each CSD (programming) and the CEU (programming) and starts each CSD and the CEU according to the serial interface. Therefore each test round gets configured and the particular test round starts (the test round itself is managed by the CEU). After a test round is completed the host computer receives the statistical data from the CEU by the serial interface. Furthermore, the host computer prepares the statistical data from each test round and therefore for each test run for the user.

5.3 Experimental Process

The experimental process is split into a design process, generation phase, run phase and analysis.

Design Process

The hardware design process is managed by the ALTERA Quartus II software. The ALTERA Quartus II software is a FPGA design software to create the hardware description from a textual description (d hardware synthesis)[Alt12]). Furthermore, the ALTERA Quartus II software includes the Altera System on a Programmable Chip (SOPC) Builder software. The SOPC builder is a software that provides the automatic connection of IP cores and the configuration of IP cores[Alt11b]. Furthermore, the architecture of the test system is build up by the SOPC builder e.g. configuration of IP core processors and connection of these processors to their associated IP cores. In addition, the overall FPGA design is managed by Quartus II. In the design process the system prototype is generated by Quartus II (hardware synthesis). Furthermore, in this phase the granularity of the global time base is set. In addition, the frequency of the test system is defined.

Generation Phase

When the hardware synthesis is completed the test runs can be performed on the hardware platform. Each test run requires data such that the number of test rounds and CSDs are defined. Each CSD has a number of MCEs in each test round and the configuration data of the CAN Controller is provided. These test data are generated by a self-written JAVA program called GEN_MSG_EXCELL that executes at the host computer. In this program the SMINT, BSTEP, RSTEP, number of test rounds (if RSTEP is zero), number of CSDs, number of MCEs, id of the RAMP MCE, location of the RAMP MCE (RAMP CSD), duration of the test round time and the configuration data of the CAN Controller have to be set. The GEN_MSG_EXCELL program generates a text file (called CAN_DATA.txt) that is used for each test round of a test run.

Run Phase

After the generation of the CAN_DATA.txt file the run phase can continue. Each run phase can be repeated for multiple times but each test run will be configured with the same test run data. Each test run is controlled by a script called linux_testrun_script.sh that executes at the host computer. In the script the start test round and the end test round have to be defined in order to determine the bandwidth interval of the test run. Furthermore, the assumed utilization of the test system in each particular test round can be observed in two EXCEL-sheets (Excel is a table calculation software) called 4nodes.xlsx (4 CSDs) and 8nodes.xlsx (8 CSDs). These two Excel sheets calculate TBAND for each test round. Moreover, the script runs in the loop that starts at the start of the test round and ends at the end of the test round.

In each test round a program called MSGATR_BUILDER executes at the host computer. The MSGATR_BUILDER-program reads the CAN_DATA.txt file (overhanded by the script) in order to chose the particular test round in the text file (the test round number is overhanded by the script) and extracts these data into the header files of the main programs of each CSD and the CEU. The header file of each CSD consists of the MCE data (id, MINT and RAND), duration of the test round (in ticks of the global time base), definition of interrupt interval of the local time base and configuration data of the CAN Controller (prescaler, PHASE_SEG1 and PHASE_SEG2 - see Subsection 3.1). In addition, the header file of the CEU consists of the time-out of the interrupt timer in order to detect stuck nodes, duration of the test round to detect RMIs which were received after the test round end, ratio of the global time base in μs and number of CSDs (to switch off unused CSDs if a particular test run consists of 4 CSDs).

The script overhands the log-directory (directory where the statistical data from each test round is stored) to the MSGATR_BUILDER-program that generates the startup-script (script that runs after the call of the command line tool, called usherbash.rc) of the Nios II Command Shell. The Nios II Command Shell is a shell that is used to manage the complete design process of the FPGA-prototype board e.g. hardware synthesis and programming of processors placed onto the FPGA[Alt11b]. Furthermore, the startup-script of the Nios II Command Shell consists of the compile command and download of each program of the CSD and of the CEU. In addition,

the startup-script defines a call of a terminal in order to listen to the CEU via the serial interface of the FPGA-prototype board to receive the statistical data of each test round. These data will be stored in a log file (one log file for each test round) in the log directory. This terminal runs for an intended time as defined in the startup-script (duration of the test round in addition to an extra time budget for completion) e.g. to calculate the statistical data. After the completion of the MSGATR_BUILDER-program the Nios II Command Shell is called by the script that runs the main loop. Furthermore, the Nios II Command Shell executes as defined in the startup-script and terminates after the reception of the statistical data (defined by the duration and the extra time budget). After the termination of the Nios II Command Shell the main loops continues at the start with the next test round or interrupts due to the end of the test run or test run interval.

Analysis

After the completion of a test run a Java program (called Preproc) is used to extract the statistical data of the logging files into Excel sheets as follows:

- "bandwidth.csv" that lists the average utilization of the transmission medium for each test round. Therefore the number of total sent TMIs are summed up, multiplied by the frame length of a CAN frame and divided by the test round duration in seconds.
- "broken.csv" that lists all erroneous test rounds.
- "msg_atr.csv" that lists the MINT, RAND, send omissions and number of send TMIs of each MCE.
- "rec_stat.csv" that lists the average transmission time, maximum transmission time, minimum transmission time, receive omissions and the variance of the transmission times of each MCE for each test round. Furthermore, the MCEs are related to their CSD.
- "sen_id_list.csv" that lists all MCE ids in respect to the CSD that used the corresponding MCE.

Furthermore, the sheets can be limited by the MCE ids as defined by the user.

5.4 Experiments

The timing model is based on event triggered messages which are periodically send in variable intervals. The messages of the timing model are artificially created in order to simulate real world applications (see Subsection 4.1). Furthermore, the transmission times of messages are important as they are hardly influencing the dead lines of processes because processes require information to fulfill[Kop97, P. 227 FF]). In general, it is impossible to model a CAN network perfectly. There are many parameters such as overload frames, arbitration, process executions that vary in time, etc. Furthermore, a test system that simulates processes which emit messages via a (real) CAN network will give accurate timings due to statistical probability.

The prototype provides the simulation of processes and each process emits a message at the end of its duration. In addition, the change of the execution time of a task during the run time is simulated by the RAND of a MCE that simulates such a process. Furthermore, the prototype provides any timely behaviour of the simulated processes and the emission of messages but not request messages. However in the current implementation the message priority is coupled to the sending frequency and thus periodic messages are used. Though it is possible to emulate sporadic messages and a particular sporadic message can be created by a high MINT in combination with a RAND that is one (n modulo 1 is always 0).

The test runs without a RAMP MCE are used to exploit the network with a typical utilization of 40%. Furthermore, test runs with a high priority MCE are used to determine the impact of overload conditions. In order to show the impact of a higher utilization the analysis of the test runs with a high priority MCE is split into bandwidth intervals (200 - 230 kbit/s, 230 - 300 kbit/s and 300 - 350 kbit/s). Furthermore, the test runs with a low priority MCE show the impact of the uninterruptedness of message transmissions.

Each test run consists of many test rounds and at the end of each test round statistical data are created (see Subsection 4.1). Thus the data volume has to be compressed to illustrate the behaviour of the network. Therefore the statistical data of each round is used to create statistical data (meta statistic) of the test run. The meta statistic constitutes as follows:

- Average send frequency of the TMIs: The emitted TMIs per MCE of each test round are summed up and divided by the number of test rounds.
- Total minimum send frequency of the TMIs: The minimum number of sent TMIs of a MCE during a test round. This attribute is determined for all test rounds.
- Total maximum send frequency of the TMIs: The maximum number of sent TMIs of a MCE during a test round. This attribute is determined for all test rounds.
- Mean deviation of the send frequencies: The mean deviation is calculated for each MCE and is made up by the following formula: $\sqrt{\frac{1}{n-1} * \sum(x_i - x)}$ where n is the number of test rounds, x_i is the number of sent TMIs of a MCE T in test round i and x is the average send frequency of MCE T.
- Average transmission time: The average of the average transmission times of of a MCE of all test rounds.
- Average maximum transmission time: The average of the maximum transmission times of a MCE of all test rounds.
- Total maximum transmission time: The denotation of the longest maximum transmission time of a MCE of all test rounds.
- Message transmission time jitter: The difference between the total maximum transmission time and the total minimum transmission time. The total minimum transmission time is the denotation of the fastest transmission time of a MCE of all test rounds.

- Total maximum transmission time of a CSD: The denotation for the longest transmission of all total maximum transmission times of a CSD.
- Deviation of the average transmission times: The mean deviation is calculated for the average transmission time of each test round by the following formula: $\sqrt{\frac{1}{n-1} * \sum(x_i - x)}$ where n is the number of test rounds, x_i denotes the average transmission time of a MCE T in test round i and x denotes the average transmission time of all test rounds of MCE T.
- Deviation of the maximum transmission times: The mean deviation is calculated for the maximum transmission time by the following formula: $\sqrt{\frac{1}{n-1} * \sum(x_i - x)}$ where n is the number of test rounds, x_i denotes the maximum transmission time of a MCE T in test round i and x denotes the average maximum transmission time of all test rounds of MCE T.
- Measured average variance of the transmission times: The average of the variances of the transmission times for each MCE of all test rounds.
- Deviation of the measured variances of the transmission times: The mean deviation is calculated for the variances of the transmission times by the following formula: $\sqrt{\frac{1}{n-1} * \sum(x_i - x)}$ where n is the number of test rounds, x_i denotes the variance of the transmission times of MCE T in test round i and x is the measured average variance of the transmission times of MCE T.

CHAPTER 6

Results

This Chapter presents the results of the test runs. Furthermore, 1024 bits equals 1 kbit. Note that the calculation of the bandwidth consumption is always based on a frame length of 131 bits and that the utilization is always higher due to bit stuffing. Furthermore, there were not receive omissions in any test run.

6.1 4 CSDs - No Ramp MCE

This Subsection presents the results obtained by the test run with 4 CSDs and a RSTEP of zero.

Network Utilization

Table 6.1 shows the average send frequency of the TMIs (mean), the total minimum send frequency of the TMIs (min), the total maximum send frequency of the TMIs (max) and the mean deviation of the send frequencies of the TMIs (deviation) of the highest and lowest priority MCE of each CSD (the Ramp CSD has a single MCE) for the complete test run.

As defined in Chapter 4 the send frequencies correlate with the priorities of the MCEs. In addition, the deviation is small and fits the value of the mean. Therefore the CAN network demonstrates a good average send behaviour. Furthermore, on average 1533 TMIs/s were totally send in each round and therefore the bandwidth utilization was 196 kbit/s. Thus the communication medium has an average load of 39.2%. Moreover, 12,261,320 TMIs were totally send during the test run. Therefore in the average 15,326 TMIs were send during each test round.

Average Transmission Times

Table 6.2 shows the average transmission time (arithmetic mean), the average maximum transmission time (average maximum), the total maximum transmission time (total maximum) and the message transmission time jitter (jitter) of the highest and lowest priority MCE of each CSD (the Ramp CSD has a single MCE).

id	mean(messages/round)	min	max	deviation(messages/round)
0x1	622.26	601	649	7.46
0x2	611.06	589	636	7.26
0x3	600.24	575	624	6.81
0x4	589.09	568	612	6.99
0x1d	408.73	386	430	5.86
0x1e	403.7	387	424	5.72
0x1f	398.73	381	415	5.67

Table 6.1: Mean, minimum, maximum and standard deviation of the send behavior of a test system with 4 CSDs and a RSTEP of zero

id	arithmetic mean(μs)	average maximum(μs)	total maximum(μs)	jitter(μs)
0x1	495.12	852.8	991	686
0x2	538.67	1591.75	3051	2746
0x3	542.93	1668.11	3051	2746
0x4	544.52	1722.4	4119	3814
0x1d	655.44	2426.9	4043	3662
0x1e	661.43	2530.55	4119	3738
0x1f	669.44	2596.73	4348	4043

Table 6.2: Mean and maximum statistics of the transmission times of a test system with 4 CSDs and a RSTEP of zero

The arithmetic mean and average maximum hardly correlate to the priority of the MCEs. Therefore a lower priority leads to a longer average transmission time but a higher priority does not guarantee that a MCE has a better total maximum and jitter than a lower priority MCE. Moreover, the priority of a MCE does not exactly reflect the order of the total maximum and jitter but a correlation exists. Furthermore, the jitter, total maximum and average maximum of MCE 0x1 are limited compared to the other MCEs. Moreover, the arithmetic mean linearly increases with the MCE priority and the average maximum raises also linearly (except MCE 0x1) but with a higher gain.

Table 6.3 shows the total maximum transmission time for each CSD (CSDR denotes the Ramp MCE) and the MCEs which generated the total maximum transmission times. The total maximum transmission time over all MCEs (worst total maximum transmission time) is 4348 μs and the total maximum transmission times are similar. Furthermore, a higher MCE priority does not guarantee a low total maximum transmission time. This is exemplified by the MCE 0x11 that has the slowest total transmission time of CSD 0. CSD 0 has 10 MCEs and MCE 0x11 has the 4th highest priority.

	CSDR	CSD0	CSD1	CSD2
MCE	0x1	0x11	0x1b	0x1f
total maximum(μs)	991	4119	4272	4348

Table 6.3: Total maximum transmission times of each CSD of a test system with 4 CSDs and a RSTEP of zero

id	mean deviation(μs)	max deviation(μs)	average var(μs)	var deviation(μs)
0x1	4.49	53.78	9751.5	594.38
0x2	5.67	313.47	18590.25	3122.31
0x3	5.93	346.62	20841.24	3698.65
0x4	6.28	389.9	22600.21	4320
0x1d	13.91	393.86	82050.13	13332.85
0x1e	14.75	415.05	90232.17	16035.4
0x1f	16.02	437.17	96308.23	17639.85

Table 6.4: Variance statistics of the transmission times of a test system with 4 CSDs and a RSTEP of zero

Deviations of the Transmission Times

Table 6.4 shows the deviation of the average transmission times (mean deviation), the deviation of the maximum transmission times (max deviation), the measured average variance of the transmission times (average var) and the deviation of the measured variances of the transmission times (var deviation) of the highest and lowest priority MCE of each CSD (the Ramp CSD has a single MCE).

The MCE priority hardly correlates with each deviation and with the average variance of the transmission times. Furthermore, a lower priority yields to a higher deviation/variance. Moreover, the max deviation of all MCEs except MCE 0x1 are close to each other. The dispersions are hardly limited for MCE 0x1 and verify the small jitter of MCE 0x1 in Table 6.2. Moreover, the dispersions of MCE 0x1 are very limited when compared to the other MCEs and the statistical values. In contrast, the mean deviation of the other MCEs are significantly increased when compared to MCE 0x1.

6.2 8 CSDs - No Ramp MCE

This Subsection presents the results obtained by the test run with 8 CSDs and a RSTEP of zero.

Network Utilization

Table 6.5 shows the average send frequency of the TMIs (mean), the total minimum send frequency of the TMIs (min), the total maximum send frequency of the TMIs (max) and the mean deviation of the send frequencies of the TMIs (deviation) of the highest and lowest priority MCE of each CSD (the Ramp CSD has a single MCE) for the complete test run.

id	mean(messages/round)	min	max	deviation(messages/round)
0x1	621.96	599	645	7.22
0x2	579.04	561	599	6.71
0x3	541.48	518	560	6.73
0x4	508.38	488	531	6.31
0x5	479.09	462	506	6.12
0x6	452.89	432	473	6.27
0x7	429.58	414	447	5.81
0x8	408.26	389	428	5.78
0x41	107.13	98	115	2.88
0x42	105.88	96	118	3.06
0x43	104.32	96	112	2.94
0x44	103.23	94	112	2.97
0x45	101.88	93	110	2.86
0x46	100.57	92	110	2.91
0x47	99.55	90	110	3.06

Table 6.5: Mean, minimum, maximum and standard deviation of the send behavior of a test system with 8 CSDs and a RSTEP of zero

As defined in Chapter 4 the send frequencies correlate with the priorities of the MCEs. In addition, the deviation is small and fits the value of the mean. Therefore the CAN network shows a good average send behaviour. Furthermore, on average 1557 TMIs/s were totally send in each round and therefore the bandwidth utilization is 199.2 kbit/s. Thus the communication medium has an average load of 39.8%. Moreover, 12,455,560 TMIs were totally send during the test run. Therefore on average 15,569 TMIs were send in each test round.

Average Transmission Times

Table 6.6 shows the average transmission time (arithmetic mean), the average maximum transmission time (average maximum), the total maximum transmission time (total maximum) and the message transmission time jitter (jitter) of the highest and lowest priority MCE of each CSD for the complete test run (the Ramp CSD has a single MCE).

The arithmetic mean and average maximum hardly correlate with the priority of the MCEs. The arithmetic mean of MCE 0x41 is slightly increasing up MCE 0x42 and the average maximum of MCE 0x3 is marginally raising up MCE 0x4. The same is true for MCE 0x7 up to MCE 0x8. However a lower priority typically leads to a longer average transmission time and the gain of the arithmetic means linearly increase with the priority of the MCEs.

Furthermore, the total maximum and therefore jitter of the MCEs are weakly correlating. For example, the total maximum of MCE 0x3, that has the third highest priority, has about the same total maximum as the lowest priority MCEs in the test run. However high priority MCEs typically have lower total maximum transmissions in contrast to low priority MCEs. Furthermore, the jitter as well as the other statistical values except for the arithmetic mean of MCE

id	arithmetic mean(μs)	average maximum(μs)	total maximum(μs)	jitter(μs)
0x1	497.59	843.74	991	686
0x2	535.25	1609.01	3966	3661
0x3	538.74	1629.32	5492	5187
0x4	539.42	1620.37	4577	4272
0x5	541.97	1639.13	3890	3585
0x6	544.55	1650.94	4806	4501
0x7	547.7	1685.78	4424	4119
0x8	549.97	1676.42	4348	4043
0x41	687.5	2456.45	4729	4348
0x42	687.09	2479.16	5950	5569
0x43	690.8	2515.57	5492	5111
0x44	692.76	2542.68	5492	5111
0x45	696.62	2558.96	5187	4806
0x46	699.32	2630.96	6789	6408
0x47	704.79	2664.9	6026	5645

Table 6.6: Mean and maximum statistics of the transmission times of a test system with 8 CSDs and a RSTEP of zero

	CSDR	CSD0	CSD2	CSD3	CSD4	CSD5	CSD6	CSD7
MCE	0x1	0x25	0x42	0x43	0x3d	0x3e	0x46	0x47
total maximum(μs)	991	5340	5950	5492	6026	5645	6789	6026

Table 6.7: Total maximum transmission times of each subscriber of a test system with 8 CSDs and a RSTEP of zero

0x1 are very limited when compared to the other MCEs and for the other MCEs these statistical values are significantly increased when compared to MCE 0x1.

Table 6.7 shows the total maximum transmission time for each CSD (CSDR denotes the Ramp CSD) and the MCEs which generated the total maximum transmission times. The worst total maximum transmission time is 6789 μs and the total maximum transmission times are similar. Furthermore, a higher MCE priority does not guarantee a low total maximum transmission time. This is shown MCE 0x25 that has the slowest total transmission time of CSD 0. CSD 0 has 10 MCEs and MCE 0x25 has the 6th highest priority. However MCE 0x25 has the fastest total maximum transmission times of all MCEs that generate the total maximum transmission times for each CSD but not the Ramp MCE.

Deviations of the Transmission Times

Table 6.8 shows the deviation of the average transmission times (mean deviation), the deviation of the maximum transmission times (max deviation), the measured average variance of the transmissions times (average var) and the deviation of the measured variances of the transmis-

id	mean deviation(μs)	max deviation(μs)	average var(μs)	var deviation(μs)
0x1	4.2	52.35	9625.35	591.16
0x2	5.23	451.68	15912.26	3819.27
0x3	6.05	524.74	17237.32	4944.44
0x4	6.24	459.22	18177.68	4294.09
0x5	6.4	453.44	19475.28	4493.92
0x6	6.84	501.25	20780.73	5448.67
0x7	7.14	484.78	22309.52	5504.46
0x8	7.45	470.47	23859.65	6178.91
0x41	33.48	562	125588.6	46334.91
0x42	35.69	607	128565.76	51685.28
0x43	36.9	614	133120.8	51638.88
0x44	35.55	633.59	136989.19	54100.41
0x45	35.68	630.54	141722	54197.74
0x46	37.86	739.15	149165.64	64633.01
0x47	38.51	678.8	152279.21	61217.02

Table 6.8: Variance statistics of the transmission times of a test system with 8 CSDs and a RSTEP of zero

sion times (var deviation) of the highest and lowest priority MCE of each CSD for the complete test run (the Ramp CSD has a single MCE).

The MCE priority tightly correlates with the mean deviation. The mean deviation of MCE 0x42 is slightly increasing up to MCEs 0x44 and 0x45 and the mean deviation of MCE 0x43 is lowly increasing up to MCEs 0x44 and 0x45. Furthermore, the mean deviation of low priority MCEs are significantly increased when compared to high priority MCEs. In addition, the max deviation exponentially increases with the MCE priority but not for MCE 0x1 that has a limited max deviation. The max deviation weakly correlates with the MCE priority. For example, the max deviation of MCE 0x3 is significantly increasing up to MCEs 0x4, 0x5, 0x6, 0x7 and 0x8 and close to the deviations of the low priority MCEs. However high priority MCEs have typically a lower max deviation than low priority MCEs. The data from MCE 0x3 confirms the results of the total maximum in Table 6.8.

The average variance as well as the var deviation hardly correlate with the priority of the MCEs. Furthermore, the dispersions (average variance and var deviation) of MCE 0x1 are very limited when compared to the other MCEs. The dispersions are hardly limited for the MCE 0x1 and verifies the small jitter of MCE 0x1 in Table 6.6. Furthermore, the average variance and the var deviation of low priority MCEs are extremely increased when compared to high priority MCEs.

6.3 4 CSDs - High Priority Ramp MCE

This Subsection presents the results obtained by the test run with 4 CSDs and with a high priority Ramp MCE.

The test run is split into intervals (bandwidth interval) and each interval has an individual RSTEP. The first interval starts with a utilization of 40% of the communication medium and the bandwidth increases for each test round with the individual RSTEP as described in Subsection 4.2. In addition, each interval ends at a defined bandwidth utilization and the next interval continues. This tendency does not apply for the last interval. Furthermore, there are totally three intervals from 200 kbit/s to 230 kbit/s (first interval), 230 kbit/s to 300 kbit/s (second interval) and 300 kbit/s to 350 kbit/s (third interval). The RSTEP is 10 μs for the first interval, 3 μs for the second interval and 1 μs for the third interval. The RSTEP for the first interval has the lowest granularity and the RSTEP for the third interval has the highest granularity because the bandwidth utilization increases exponentially. An exponential increased utilization leads in conjunction with a high RSTEP to an improper analysis for higher test rounds. Therefore, an exponentially increased utilization demands a low RSTEP for high test rounds but unnecessarily raises the number of test rounds at low utilizations. Thus splitting up into intervals with suitable individual RSTEPs yields to a detailed analysis and reduces the number of test rounds for the total test run. Therefore the tables in this Subsection are divided into the three intervals (int 1, int 2, int 3) such that each part refers to an interval.

Network Utilization

The first interval derives from a test run consisting of 800 test rounds with a RSTEP of 10 μs. Figure 6.1 shows the increasing send behavior of the Ramp MCE (MCE 0x1) with the number of sent TMIs and the send omissions of the Ramp MCE over the test rounds of the test run consisting of 800 test rounds and a RSTEP of 10 μs. Furthermore, the x-axis shows the round number and the y-axis shows the numbers of the send omissions (node_0_1_omissions) and sent TMIs (node_0_1_send_total) for each test round with a duration of 10 seconds. The maximum send capacity of the Ramp MCE is 3458 TMIs/s at test round 789 and at test round 789 all other MCEs are blocked due to the Ramp MCE that has the highest priority. Thus the total maximum bandwidth has 442.4 kbit/s (88.5% of 500 kbit/s). Note that the number of emitted TMIs fluctuates with a full load of the communication medium. Furthermore, the send omissions for the Ramp MCE is limited to 6552 messages per seconds due to the granularity of the local time base that triggers the TMIs. Furthermore, single send omissions happen before the full load of the communication medium.

Figure 6.2 shows the emission behaviour of the highest and lowest MCE of each CSD but not the RAMP CSD with the number of sent TMIs over the test rounds of the test run consisting of 800 test rounds and a RSTEP of 10 μs. Furthermore, the x-axis shows the round number and the y-axis shows the numbers of the sent TMIs (node_x_p_send_total where x denotes the CSD and by p the id of the MCE that is sent by CSD x) for each test round with a duration of 10 seconds. The send behavior of the CSDs are stable until the Ramp MCE cuts off (a cut off means that all MCEs but not the Ramp MCE do not emit TMIs or at least limitedly emits TMIs) all other MCEs. The send omissions of MCE 0x1f starts to raise at test round 777. At test run 777 the measured bandwidth consumption is 453 kbit/s or 90.6% of 500 kbit/s. Furthermore, the MCE priority correlates with the send frequency as defined in Subsection 4.2. Moreover, a

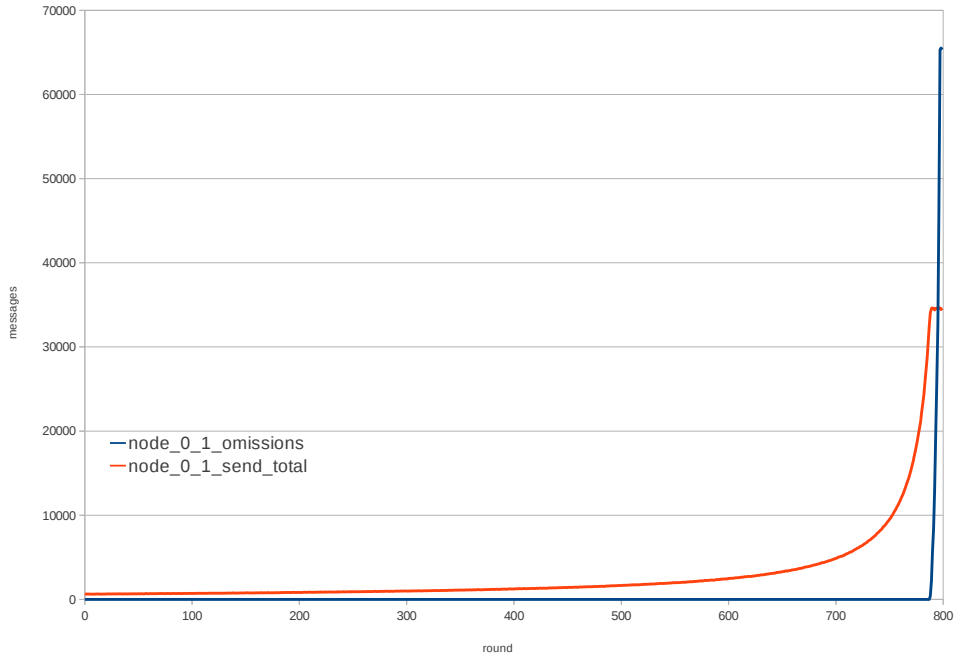


Figure 6.1: Total sent TMIs and send omissions of MCE 0x1 in the test system with 4 CSDs and a high priority Ramp MCE

high priority MCE will be cut off later than a low priority MCE. However in some test rounds with high utilization of the communication medium there are high priority MCE cut offs and low priority MCEs emissions.

Table 6.9 shows the average send frequency of the TMIs (mean), the total minimum send frequency of the TMIs (min), the total maximum send frequency of the TMIs (max) and the mean deviation of the send frequencies of the TMIs (deviation) with the highest and lowest priority MCE of each CSD (the Ramp CSD has a single MCE) up to test round 777 of the test run consisting of 800 test rounds with a RSTEP of 10 μs. The send statistics are calculated up to test round 776 because at test round 777 the lowest priority MCE starts to produce send omissions and send omissions denote an overloaded communication medium.

The Ramp MCE (MCE 0x1) has a high mean as well as a high max because the emission of TMIs increases exponentially over the test rounds. Furthermore, as defined in Chapter 4 the send frequencies correlate with the priority of the MCEs.

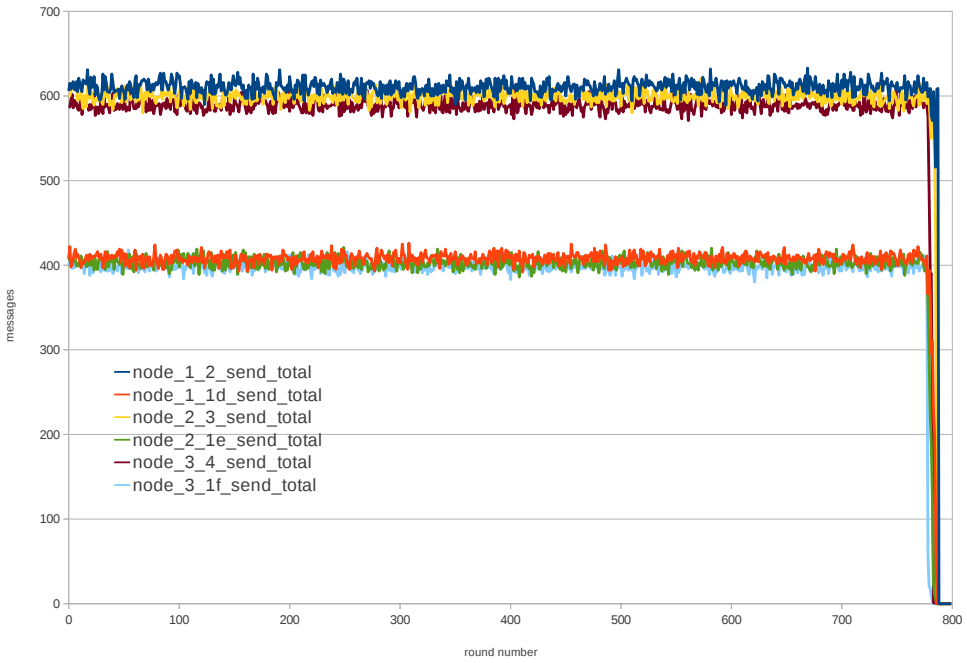


Figure 6.2: Total sent TMIs of MCEs 0x2, 0x3, 0x4, 0x1d, 0x1e, 0x1f in the test system with 4 CSDs and a high priority Ramp MCE

id	mean(messages/round)	min	max	deviation(messages/round)
0x1	2206.56	614	18904	2667.94
0x2	610.88	589	633	7.47
0x3	599.69	580	620	6.71
0x4	589.32	571	609	6.8
0x1d	408.19	392	426	5.7
0x1e	403.26	386	421	6.16
0x1f	398.83	380	418	5.66

Table 6.9: Mean, minimum, maximum and standard deviation of the send behavior of a test system with 4 CSDs and a high priority Ramp MCE

Average Transmission Times

In Figure 6.3 the total maximum and average transmission times of the highest (Ramp MCE) and lowest (MCE 0x1f) priority MCE are compared during the test rounds of the test run consisting of 800 test rounds and a RSTEP of 10 μs (test rounds 0 to 750). Furthermore, the x-axis shows the round number. In addition, the y-axis shows the total maximum transmission times of the RAMP MCE (node_1_1_max) as well as the MCE 0x1f (node_0_1f_max) in μs and the averages transmission times for the RAMP MCE (node_1_1_average) as well as the MCE 0x1f (node_0_1f_average) in μs. The total maximum and average transmission times of the Ramp

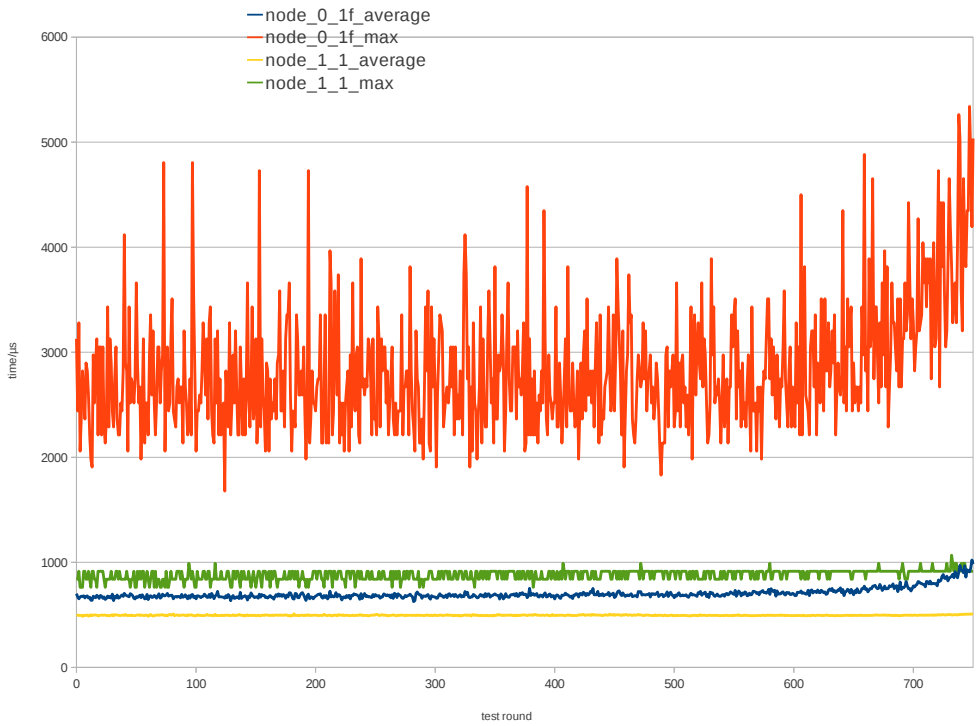


Figure 6.3: Total maximum and average transmission times of MCE 0x1 and MCE 0x1f in the test system with 4 CSDs and a high priority Ramp MCE

MCE are smoothed. In contrast, the MCE 0x1f hardly fluctuates during the test rounds. Furthermore, at test round 600 the total maximum transmission times of the MCE 0x1f increase exponentially and the average transmission times of MCE 0x1f start to increase exponential at test round 680. The total maximum transmission times as well as the average transmission times of the Ramp MCE stay constant until test round 788 (test round 788 is not shown in the figure). Moreover, the measured bandwidth consumption is 226 kbit/s in test round 600, 247 kbit/s in test round 680 and 448 kbit/s in test round 788.

Table 6.10 shows the average transmission time of each interval (arithmetic mean), the av-

erage maximum transmission time of each interval (average maximum), the total maximum transmission time of each interval (total maximum) and the message transmission time jitter of each interval (jitter) with the highest and lowest priority MCE of each CSD (the Ramp CSD has a single MCE). Table 6.10 is divided into the three intervals. Furthermore, the first interval starts at 196 kbit/s and ends at 230 kbit/s. The second interval starts at 230 kbit/s and ends at 287 kbit/s. The third interval starts at 300 kbit/s and ends at 350 kbit/s.

The statistical data for each MCE but not for the Ramp MCE increases in each interval. The arithmetic mean and average maximum increases linearly with the priority of the MCE up to 300 kbit/s. Furthermore, in the interval between 200 - 230 kbit/s the arithmetic mean as well as the average maximum hardly correlate with the MCE priority. Moreover, the total maximum and therefore the jitter have a weak correlation with the MCE priority. For example, MCE 0x3 has a total maximum of 4500 μs and is therefore higher than the total maximum of 4272 μs of MCE 0x1d. Furthermore, the arithmetic mean as well as the average maximum in the interval between 230 - 300 kbit/s hardly correlate with the priority of the MCEs. The arithmetic mean and the average maximum in these intervals are only slightly increasing up to the interval between 200 - 230 kbit/s. Moreover, the total maximum and therefore the jitter in the interval between 230 - 300 kbit/s correlate with the priority of the MCEs and are distinctly increasing up to the total maximum of the interval between 200 - 230 kbit/s.

In the interval between 300 - 350 kbit/s the arithmetic mean as well as the average maximum hardly correlate with the priority of the MCEs but they are hardly increased when compared to the other intervals and especially to the interval between 200 - 230 kbit/s. For example, the arithmetic mean of MCE 0x1f is increased by about 49% and the average maximum is increased by about 80% to the MCE 0x1f in the interval between 200 - 230 kbit/s. Furthermore, total maximum and therefore the jitter do not correlate with the MCE priority at all. In addition, the total maximum transmission times when compared to the interval between 200 - 230 kbit/s is highly increased e.g. the total maximum of MCE 0x2 is increased by about 120%.

Table 6.11 shows the total maximum transmission time for each CSD (CSDR denotes the Ramp MCE) and the MCEs which generated the total maximum transmission times for each interval. The worst total maximum transmission time is 5187 μs in interval 1, 6026 μs in interval 2 and 9307 μs in interval 3. Furthermore, a higher MCE priority does not guarantee a low total maximum transmission time. This is represented by MCE 0x9 that has the slowest total transmission time of CSD 1 in interval 1. CSD 1 has 10 MCEs and MCE 0x11 has the 3rd highest priority. Moreover, at higher load the low priority MCEs have the longest total maximum transmission times.

Deviations of the Transmission Times

Figure 6.4 shows the measured variance of the Ramp MCE (MCE 0x1) and the MCE 0x1f during the test rounds of the test run consisting of 800 test rounds and a RSTEP of 10 μs (test rounds 0 to 750). Furthermore, the x-axis shows the round number and the y-axis shows the measured variance of the RAMP MCE (node_1_1_variance) as well as the lowest priority MCE 0x1f (node_0_1f_variance) in μs. The variance of the Ramp MCE is stable and only fluctuating within a very limited range. In contrast, the variance of MCE 0x1f increases exponentially after

id	arithmetic mean(μs)	average maximum(μs)	total maximum(μs)	jitter(μs)
int 1	200 - 230 kbit/s			
0x1	495.56	874.43	991	686
0x2	546.52	1677.86	3356	3051
0x3	550.87	1737.08	4500	4195
0x4	552.98	1807.54	4195	3890
0x1d	670.81	2527.99	4272	3891
0x1e	676.09	2598.31	4958	4577
0x1f	687.04	2728.47	4806	4425
int 2	230 - 300 kbit/s			
0x1	499.3	919.61	1068	763
0x2	590.35	2058.51	3585	3280
0x3	598.26	2212.14	4577	4272
0x4	600.1	2254.12	4577	4272
0x1d	764.78	3112.14	4882	4501
0x1e	772.91	3233.26	5797	5416
0x1f	788.43	3397.35	6026	5645
int 3	300 - 350 kbit/s			
0x1	514.41	935.51	1068	763
0x2	690.04	3280.3	7399	7094
0x3	702.97	3413.58	6789	6484
0x4	718.65	3706.77	6408	6103
0x1d	1016.59	4626.03	7476	7171
0x1e	1046.98	5013.05	9307	9002
0x1f	1096.33	5379.56	8239	7858

Table 6.10: Mean and maximum statistics of the transmission times of a test system with 4 CSDs and a high priority MCE

	CSDR	CSD0	CSD1	CSD2
int 1	200 - 230 kbit/s			
MCE	0x1	0x1d	0x9	0x1f
total maximum(μs)	991	4272	5187	4806
int 2	230 - 300 kbit/s			
MCE	0x1	0x1d	0x1e	0x1f
total maximum(μs)	1068	4882	5797	6026
int 3	300 - 350 kbit/s			
MCE	0x1	0x1d	0x1e	0x1c
total maximum(μs)	1068	7476	9307	9078

Table 6.11: Total maximum transmission times of each CSD of a test system with 4 CSDs and a high priority Ramp MCE

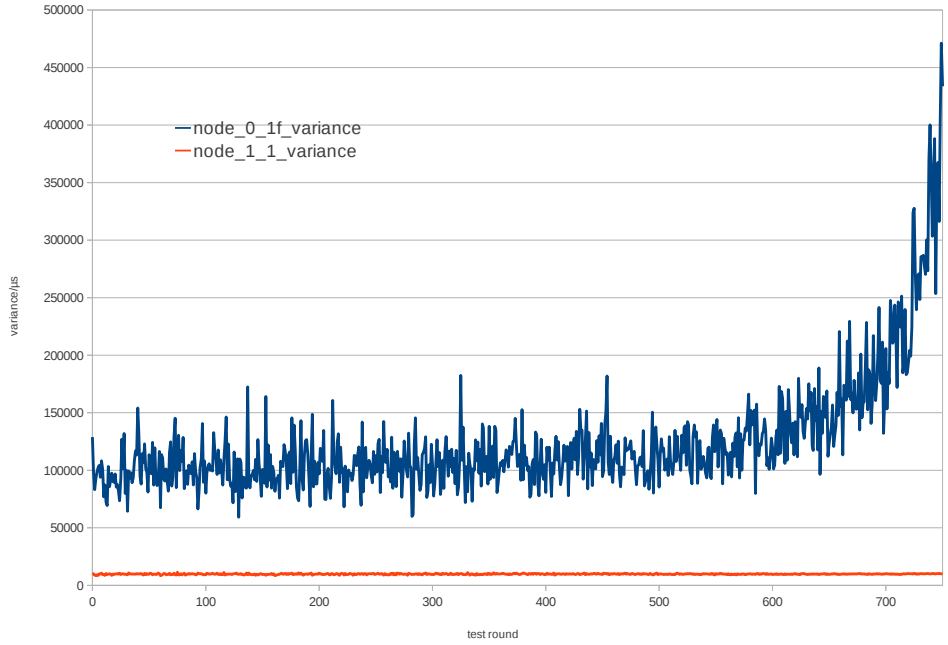


Figure 6.4: Measured variances of MCEs 0x1 and 0x1f in the test system with 4 CSDs and a high priority Ramp MCE

test round 600 and has a high fluctuation. The variance of MCE 0x1f nearly doubles from test round 600 to 700 and from test round 700 to 750. The measured bandwidth consumption is 226 kbit/s at test round 600, 258 kbit/s at test round 700 (14% higher than 226 kbit/s) and 319 kbit/s at test round 750 (24% higher than 258 kbit/s).

Table 6.12 shows the deviation of the average transmission times of each interval (mean deviation), the deviation of the maximum transmission times of each interval (max deviation), the measured average variance of the transmission times of each interval (average var) and the deviation of the measured variances of the transmission times of each interval (var deviation) with the highest and lowest priority MCE of each CSD (the Ramp CSD has a single MCE). Table 6.12 is divided into three intervals and relates to Table 6.10 (same bandwidth utilization).

The mean deviation of the RAMP MCE stays stable in all three intervals. Furthermore, the max deviation of the RAMP MCE even reduces from interval 1 to intervals 2 and 3 and is stable at the intervals 2 and 3. In addition, the average variance of the Ramp MCE slightly increases in the intervals. Moreover, the var deviation decreases for the Ramp MCE.

The mean deviation hardly correlates with the priority of the MCEs and increases for each MCE but not the RAMP MCE in the intervals. For example, the mean deviation of MCE 0x1f in-

creases from the first interval to the second interval by about 132% and from the second interval to the third interval by about 124%.

The max deviation correlates with the MCE priority. There is a violation by MCE 0x1d in interval 3 that has a lower max deviation than MCEs 0x4, 0x3 and 0x2. Furthermore, the increase of the max deviation is increased for each MCE but not the Ramp MCE in each interval.

The average variance hardly correlates with the MCE priority and increases exponentially with the priorities of the MCEs within each interval. Furthermore, the average variance hardly increases with a higher utilization of the communication medium (but not for the Ramp MCE). For example, the average variance of MCE 0x1f increases about 79% from interval 1 to 2 and about 170% from interval 2 to 3.

The var deviation hardly correlates with the priority of the MCEs and increases exponentially with the priorities of the MCEs within each interval. Furthermore, the var deviation hardly increases with a higher utilization of the communication medium (but not for the Ramp MCE). For example, the var deviation of MCE 0x1f increases about 157% from interval 1 to 2 and about 246% from interval 2 to 3.

6.4 8 CSDs - High Priority Ramp MCE

This Subsection presents the results obtained by the test run with 8 CSDs and with a high priority Ramp MCE.

The data evaluation of the test run is split into intervals. Furthermore, there are totally three intervals from 200 kbit/s to 230 kbit/s (first interval), 230 kbit/s to 300 kbit/s (second interval) and 300 kbit/s to 350 kbit/s (third interval). Therefore the tables in this Subsection are divided into the three intervals such that each part refers to an interval.

Network Utilization

The maximum send capacity of the Ramp MCE (MCE 0x1) is about 3440 TMIs per second at test round 7881 and at test round 7881 all other MCEs are cut off due to the Ramp MCE that has the highest priority. Thus the total maximum bandwidth is about 440.1 kbit/s (88.0% of 500 kbit/s). Therefore the total maximum achievable bandwidth (payload only) is 220.2 kbit/s. Note that the number of emitted TMIs fluctuates when the communication medium was fully loaded. Furthermore, the send omissions for the Ramp MCE is limited to 6578 messages per seconds due to the granularity of the local time base that triggers the TMIs. Furthermore, single send omissions happen before the communication medium is fully loaded.

The send capacity of MCE 0x47 that has the lowest priority is firstly reduced at test round 7770 with a total network load of 450 kbit/s. Note that some single send omissions happens before. In addition, the other MCEs are cut off by the Ramp MCE in further test rounds because the Ramp MCE has the highest priority.

id	mean deviation(μs)	max deviation(μs)	average var(μs)	var deviation(μs)
int 1	200 - 230 kbit/s			
0x1	3.31	47.82	9808.2	448.62
0x2	9.13	358.43	21367.53	4573.87
0x3	9.46	387.44	23445.11	5070.35
0x4	9.86	403.28	25955.68	5253.51
0x1d	20.64	392.74	93427.53	17805.07
0x1e	21.68	462.74	100401.32	20575.58
0x1f	21.24	483.03	110009.54	21725.43
int 2	230 - 300 kbit/s			
0x1	2.96	33.36	9968.13	270.37
0x2	19.14	448.12	36556.5	9608.25
0x3	19.5	541.28	41863.44	11814.24
0x4	19.7	523.8	44769.06	11557.71
0x1d	43.29	537.78	163316.26	43810.84
0x1e	45.81	616.7	178139.72	50127.48
0x1f	49.18	681.96	197257.93	55918.92
int 3	300 - 350 kbit/s			
0x1	3.92	36.66	10341.03	151.52
0x2	32.62	780.36	96033.5	31589.68
0x3	35.13	866.08	110520.12	37772.15
0x4	39.2	871.33	134685.38	46313.41
0x1d	90.53	767.71	456071.08	133900.66
0x1e	93.43	959.88	524870.3	152207.31
0x1f	110.35	936.63	629746.53	193623.65

Table 6.12: Variance statistics of the transmission times of a test system with 4 CSDs and a high priority Ramp MCE

Table 6.13 shows the average send frequency of the TMIs (mean), the total minimum send frequency of the TMIs (min), the total maximum send frequency of the TMIs (max) and the mean deviation of the send frequencies of the TMIs (deviation) of the highest and lowest priority MCE of each CSD up to test round 7769 (the Ramp CSD has a single MCE). The send statistics are calculated up to test round 7769 as in test round 7770 the lowest MCE starts to produce send omissions and send omissions denote an overloaded communication medium.

The Ramp MCE has a high mean as well as a high max as the emission of TMIs increases exponentially in the test rounds. Furthermore, as defined in Chapter 4 the send frequencies correlate with the priority of the MCEs.

Average Transmission Times

The total maximum and average transmission times of the highest (Ramp MCE) and lowest (MCE 0x47) priority MCE are compared during the test rounds of the test run in Figure 6.5

id	mean(messages/round)	min	max	deviation(messages/round)
0x1	2218	605	19665	2699.24
0x2	579.02	555	605	6.96
0x3	541.37	518	566	6.75
0x4	508.29	484	533	6.44
0x5	478.89	456	502	6.34
0x6	452.81	426	474	6.12
0x7	429.44	407	453	6.03
0x8	408.36	386	430	5.89
0x41	107.26	97	119	3.03
0x42	105.88	94	120	3.06
0x43	104.48	93	116	2.96
0x44	103.17	93	114	2.93
0x45	101.91	91	114	2.98
0x46	100.69	89	113	2.91
0x47	99.45	89	111	2.88

Table 6.13: Mean, minimum, maximum and standard deviation of the send behavior of a test system with 8 CSDs and a high priority Ramp MCE

(test rounds 0 to 7500). Furthermore, the x-axis shows the round number. In addition, the y-axis shows the total maximum transmission times of the RAMP MCE (node_1_1_max) as well as the MCE 0x47 (node_0_47_max) in μs and the averages transmission times for the RAMP MCE (node_1_1_average) as well as the MCE 0x47 (node_0_47_average) in μs. The total maximum and average transmission times of the Ramp MCE are smoothed up to MCE 0x47 which fluctuates during the test rounds. Furthermore, at about test round 6000 the total maximum transmission times of the MCE 0x1f increases exponentially and the average transmission times of MCE 0x1f starts to increase exponentially at test round 6800. The total maximum as well as the average transmission times of the Ramp MCE stay constant until test round 7779 (test round 7779 is not shown in the figure). Moreover, the measured bandwidth consumption is in test round 6000 230 kbit/s, in test round 6800 251 kbit/s and in test round 7779 452 kbit/s.

Table 6.14 shows the average transmission time of each interval (arithmetic mean), the average maximum transmission time of each interval (average maximum), the total maximum transmission time of each interval (total maximum) and the message transmission time jitter of each interval (jitter) of the highest and lowest priority MCE of each CSD (the Ramp CSD has a single MCE).

The transmission times for each MCE but not for the Ramp MCE increases in each interval. The arithmetic mean and average maximum increases linearly up to 300 kbit/s. Furthermore, in the interval between 200 - 230 kbit/s the arithmetic mean as well as the average maximum hardly correlates with the MCE priority. Moreover, the total maximum and therefore the jitter do not correlate with the MCE priority. Furthermore, the arithmetic mean as well as the average maximum in the interval between 230 - 300 kbit/s hardly correlates with the priority of the MCEs.

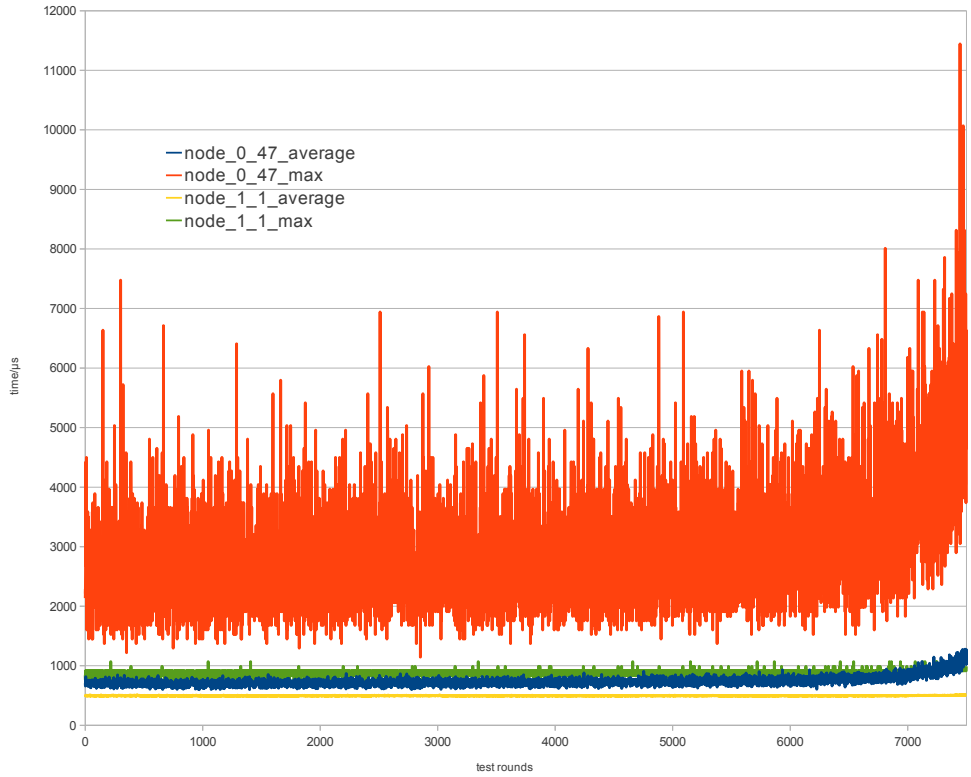


Figure 6.5: Total maximum and average transmission times of MCE 0x1 and MCE 0x46 in the test system with 8 CSDs and a high priority Ramp MCE

The arithmetic mean and the average maximum in this intervals are only slightly increased up to the interval between 200 - 230 kbit/s. Moreover, the total maximum and therefore the jitter in the interval between 230 - 300 kbit/s do not correlate with the priority of the MCEs and are increased when compared to the total maximum of the interval between 200 - 230 kbit/s.

In the interval between 300 - 350 kbit/s the arithmetic mean as well as the average maximum hardly correlate with the priority of the MCEs but they are hardly increased when compared to the other intervals. For example, the arithmetic mean of MCE 0x1f is increased by about 60% and the average maximum of MCE 0x1f is increased by about 116% to the MCE 0x1f in the interval between 200 - 230 kbit/s. Furthermore, the total maximum and therefore the jitter do not correlate with the MCE priority at all. In addition, the total maximum transmission times are highly increased when compared to the interval between 200 - 230 kbit/s e.g. the total maximum of MCE 0x2 is increased by about 69%.

Table 6.15 shows the total maximum transmission time for each CSD (CSDR denotes the Ramp MCE) and the MCEs which generated the total maximum transmission times for each interval. The total maximum transmission time in all MCEs is 8162 μs in interval 1, 9993 μs in interval 2 and 17241 μs in interval 3. Furthermore, a higher MCE priority does not guarantee a

id	arithmetic mean(μs)	average maximum(μs)	total maximum(μs)	jitter(μs)
int 1	200-230 kbit/s			
0x1	496.12	869.39	1068	763
0x2	540.01	1657.86	5950	5645
0x3	543.8	1665.07	4653	4348
0x4	544.78	1688.77	6560	6255
0x5	547.47	1701.64	5569	5264
0x6	550.18	1712.54	6103	5798
0x7	553.73	1738.14	4882	4577
0x8	555.44	1755.35	5416	5111
0x41	700.89	2547.89	6026	5721
0x42	700.92	2584.61	6637	6332
0x43	703.62	2610.38	6484	6179
0x44	707.45	2644.41	8162	7857
0x45	710.02	2672.96	6942	6637
0x46	712.08	2714.79	6560	6255
0x47	717.45	2731.28	7476	7171
int 2	230-300 kbit/s			
0x1	498.28	913.58	1068	763
0x2	577.81	2136.67	6789	6484
0x3	583.7	2168.9	6255	5950
0x4	585.65	2205.95	6865	6560
0x5	589.86	2233.2	9993	9688
0x6	594.35	2206.51	8315	8010
0x7	598.38	2246.76	7933	7628
0x8	601.3	2334.86	6713	6408
0x41	816.15	3290.69	8239	7858
0x42	816.75	3324	7552	7247
0x43	824.72	3401.38	8315	8010
0x44	830.78	3457.09	9459	9154
0x45	836.39	3567.14	8849	8544
0x46	838.89	3519.97	8620	8315
0x47	848.09	3649.07	8315	8010
int 3	300-350 kbit/s			
0x1	512.24	926.81	1068	763
0x2	659.07	3720.05	8773	8468
0x3	668.06	3775.52	7399	7094
0x4	674.71	3946.5	9612	9307
0x5	681.76	3848.13	11290	10985
0x6	693.25	4041.07	10298	9993
0x7	699.51	3944.04	8696	8391
0x8	707.66	4164.74	10604	10299
0x41	1166.74	5734.58	10832	10451
0x42	1186.68	5768.03	17241	16860
0x43	1200.83	5963.04	15028	14647
0x44	1209.77	6045.43	11672	11367
0x45	1218.03	6250.21	14113	13732
0x46	1247.91	6269.38	12434	12053
0x47	1266.89	6457.06	12358	11977

Table 6.14: Mean and maximum statistics of the transmission times of a test system with 8 CSDs and a high priority MCE

	CSDR	CSD0	CSD1	CSD2	CSD3	CSD4	CSD5	CSD6
int 1								
MCE	0x1	0x3a	0x42	0x3c	0x44	0x45	0x46	0x47
total maximum(μs)	1068	6103	6637	6713	8162	6942	6560	7476
int 2								
MCE	0x1	0x41	0x42	0x43	0x5	0x29	0x46	0x47
total maximum(μs)	1068	8239	7552	8315	9993	9001	8620	8315
int 3								
MCE	0x1	0x41	0x42	0x43	0x44	0x45	0x38	0x39
total maximum(μs)	1068	10832	17241	15028	11672	14113	13731	12434

Table 6.15: Total maximum transmission times of each CSD of a test system with 8 CSDs and a high priority Ramp MCE

low total maximum transmission time.

Deviations of the Transmission Times

Figure 6.6 shows the measured variance of the Ramp MCE and the MCE 0x47 from test round 0 to 7500. Furthermore, the x-axis shows the round number and the y-axis shows the measured variance of the RAMP MCE (node_1_1_variance) as well as the lowest priority MCE 0x47 (node_0_47_variance) in μs. The variance of the Ramp MCE is stable and only fluctuating within a very limited range. Furthermore, the variance MCE 0x47 extremely fluctuates and increases exponentially after test round 6200. The measured bandwidth consumption is at test round 6200 233 kbit/s.

Table 6.16 shows the deviation of the average transmission times of each interval (mean deviation), the deviation of the maximum transmission times of each interval (max deviation), the measured average variance of each interval (average var) and the deviation of the measured variances of the transmission times of each interval (var deviation) of the highest and lowest priority MCE of each CSD (the Ramp CSD has a single MCE). Furthermore, the table is divided into three intervals.

The mean deviation hardly correlates with the priority of the MCEs and increases for each MCE but not the RAMP MCE in each interval. For example, the mean deviation of MCE 0x47 increases from the first interval to the second interval by about 93% and from the second interval to the third interval by about 89%.

The max deviation correlates with the MCE priority. There are just few violations. The max deviation is hardly increased for each MCE but not fir the Ramp MCE in each interval. For example, MCE 0x47 has an increased max deviation of 42% from interval 1 to 2 and 65% from interval 2 to 3. Furthermore, the max deviation decreases from interval 1 to intervals 2 and 3 by 67%.

The average variance hardly correlates with the MCE priority and exponentially increases for the MCEs in each interval. Furthermore, the average variance hardly increases with a higher

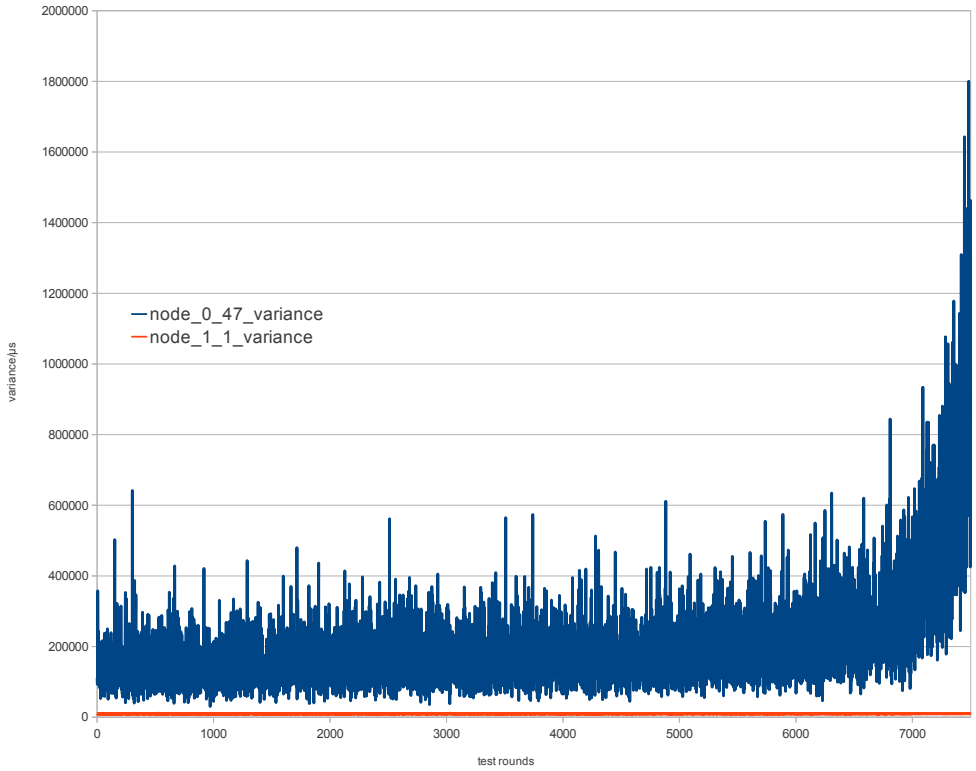


Figure 6.6: Measured variances of MCEs 0x1 and 0x47 in the test system with 8 CSDs and a high priority Ramp MCE

utilization of the communication medium (but not for the Ramp MCE). For example, the average variance of MCE 0x47 increases about 80% from interval 1 to 2 and about 250% from interval 2 to 3. Furthermore, the average variance of the Ramp MCE only slightly increases within the intervals.

The var deviation hardly correlates with the priority of the MCEs and exponentially increases for the MCEs in each interval. Furthermore, the var deviation hardly increases with a higher utilization of the communication medium (but not for the Ramp MCE). For example, the var deviation of MCE 0x1f increases about 170% from interval 1 to 2 and about 200% from interval 2 to 3. Furthermore, the var deviation of the Ramp MCE decreases from interval 1 to 2 by 58% and from interval 2 to 3 by 61%.

6.5 4 CSDs - Low Priority Ramp MCE

This Subsection presents the results obtained by the test run with 4 CSDs and with a low priority Ramp MCE.

id1	mean deviation(μs)	max deviation(μs)	average var(μs)	var deviation(μs)
int 1	200-230 kbit/s			
0x1	3.5	48.17	9565.88	441.43
0x2	7.76	486.08	17349.77	4505.99
0x3	8.26	476.18	18815.28	4721.24
0x4	8.57	493.99	20187.08	5348.47
0x5	9.03	506.83	21626.32	5979.19
0x6	9.53	497.95	23106.96	6307.48
0x7	9.96	494.18	24820.46	6731.68
0x8	10.36	499.7	26272.26	7219.88
0x41	39.51	632.94	138459.1	55309.18
0x42	39.99	645.26	143560.47	57094.25
0x43	40.81	655.08	148007.51	59600.23
0x44	41.33	660.64	153730.29	62472.1
0x45	42.55	685.85	158291.93	65506.36
0x46	43.54	702.02	162907.83	68076.58
0x47	45.02	716.66	167216.75	71000.64
int 2	230-300 kbit/s			
0x1	4.28	30.51	9663.87	268.15
0x2	20.03	736.58	30703.17	12016.05
0x3	20.97	746.22	33355.48	12576.13
0x4	22.01	779.75	36297.51	14568.76
0x5	22.63	789.43	39031.16	16523.52
0x6	23.92	770.09	41607.98	16692.43
0x7	24.27	779.04	44148.23	17481.76
0x8	25.53	846.2	48010.12	20789.76
0x41	84.78	958.64	272065.11	139744.78
0x42	85.54	940.68	278413.86	139174.62
0x43	85.98	953.16	294672.76	144163.89
0x44	89.54	1001.44	305774.65	155856.35
0x45	94.19	1054.24	323810.09	171665.06
0x46	93.96	1021.65	327265.72	173775.68
0x47	97.12	1094.97	348035.48	191457.56
int 3	300-350 kbit/s			
0x1	4.47	31.87	9992.36	172.02
0x2	27.91	1239.69	85786.24	39402.5
0x3	30.06	1124.67	91838.44	38496.61
0x4	34.13	1460.86	106219.11	58357.61
0x5	38.42	1445.43	110884.68	65212.29
0x6	38.04	1602.53	124688.31	65333.33
0x7	41	1520.56	131608.11	73334.22
0x8	43.14	1535.45	144161.8	73873.8
0x41	156.36	1518.91	949824.34	414966.4
0x42	170.72	1674.35	1014729.73	505315.69
0x43	183.68	1827.95	1093626.06	586314.8
0x44	189.48	1670.29	1102585.87	576198.34
0x45	188.35	1995.4	1186350.84	647885.09
0x46	187.95	1738.16	1230219.06	597459.44
0x47	198.82	1873.92	1314700.86	675626.24

Table 6.16: Variance statistics of the transmission times of a test system with 8 CSDs and a high priority Ramp MCE

id	mean(messages/round)	min	max	deviation(messages/round)
0x1	621.91	591	644	7.37
0x2	610.97	584	632	7.33
0x3	599.77	574	623	7.18
0x1c	413.27	395	434	5.92
0x1d	408.25	388	428	5.82
0x1e	403.38	386	423	5.93
0x1f	1912.24	391	19850	3319.92

Table 6.17: Mean, minimum, maximum and standard deviation of the send behavior of a test system with 4 CSDs and a low priority Ramp MCE

Network Utilization

The maximum send capacity of the Ramp MCE is 1982 TMIs per second and the communication medium has a full utilization starting at test round 1228. At test round 1228 the bandwidth consumption is 457 kbit/s or 91.4% of 500 kbit/s. The send behaviour of the CSDs is stable (excluding the Ramp MCE) during the complete test run and the Ramp MCE does not cut off any MCE because the Ramp MCE has the lowest priority (0x1f). Furthermore, the send omissions of the Ramp MCE 0x1f starts to raise at test round 1227. At test run 1227 the measured bandwidth consumption is 455 kbit/s or 91.0% of 500 kbit/s.

Table 6.17 shows the average send frequency of the TMIs (mean), the total minimum send frequency of the TMIs (min), the total maximum send frequency of the TMIs (max) and the mean deviation of the send frequencies of the TMIs (deviation) of the highest and lowest priority MCE of each CSD (the Ramp CSD 0 only has a single MCE) of the complete test run.

The Ramp MCE (MCE 0x1f) has a high mean as well as a high max because the emission of TMIs increases exponentially during the test rounds. Furthermore, as defined in Chapter 4 the send frequencies correlate with the priority of the MCEs.

Average Transmission Times

The total maximum and average transmission times of all MCEs are stable until the end of the test run. Figure 6.7 shows the average as well as total maximum transmission times of the second lowest (MCE 0x1e) and highest (MCE 0x1) priority MCE for the test rounds 1000 to 1250 in μs. Furthermore, the x-axis shows the round number. The y-axis shows the total maximum transmission times of MCE 0x1 (node_0_1_max) as well as MCE 0x1e (node_0_1e_max) in μs and the average transmission times of MCE 0x1 (node_0_1_average) as well as MCE 0x1e (node_0_1e_average) in μs. The maximum and average transmission times of all MCEs are solid until test round 1180. At test round 1180 the average and maximum transmission times of MCEs but not the Ramp MCE slightly increases. This is shown by the average and total transmission times of MCEs 0x1 and 0x1e in Figure 6.7. Furthermore, the transmission times of the Ramp MCE raises extremely at the end of the test round due to the full utilization of the communication medium and therefore a fully filled PSQ of the Ramp CSD. Moreover, the

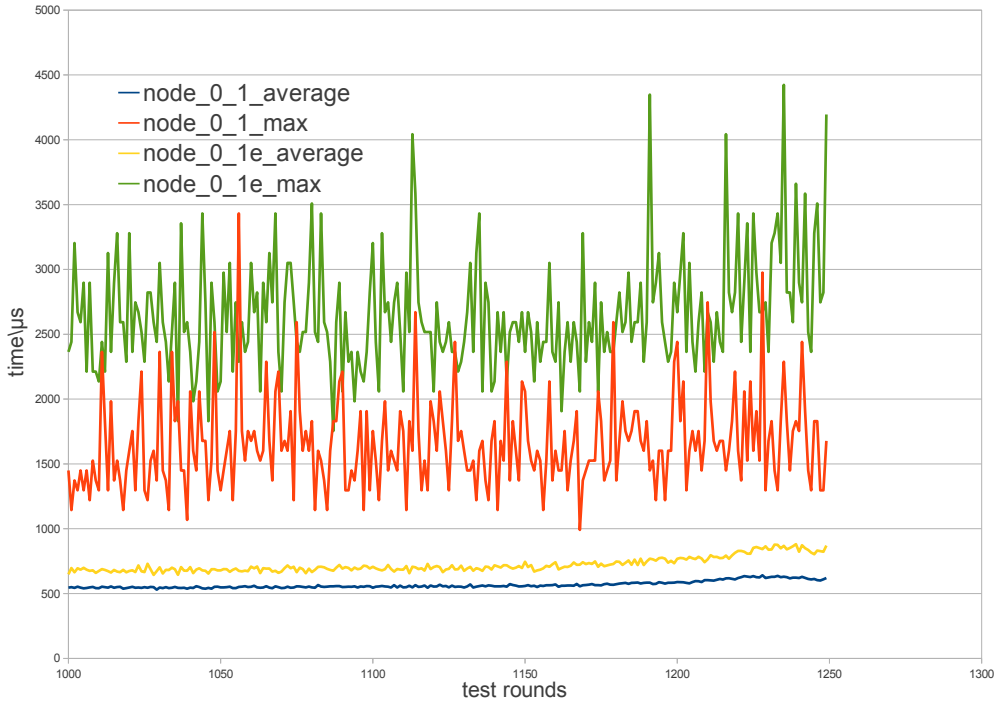


Figure 6.7: Total maximum and average transmission times of MCE 0x1 and MCE 0x1e in the test system with 4 CSDs and a low priority Ramp MCE

bandwidth consumption at test round 1180 is 287 kbit/s or 57.4% of 500 kbit/s.

Table 6.18 shows average transmission time (arithmetic mean), the average maximum transmission time (average maximum), the total maximum transmission time (total maximum) and the message transmission time jitter of each interval (jitter) of the highest and lowest priority MCE of each CSD for the complete test run (the Ramp CSD has a single MCE).

The arithmetic mean hardly correlates with the priority of the MCEs and linearly increases with the MCE priority but not for the Ramp MCE. The Ramp MCE has a clearly increased arithmetic mean when compared to the others.

The average maximum hardly correlates with the priority of the MCEs and apparently increases by the priority of the MCEs. The increase of the average maximum applies especially only the Ramp MCE.

The total maximum and therefore the jitter weakly correlates with the priority of the MCEs. Furthermore, the total maximum and jitter is extremely raised for the Ramp MCE due to the full utilization of the communication medium at the end of the test run.

Table 6.19 shows the total maximum transmission time for each CSD (CSDR denotes the Ramp MCE) and the MCEs which generates the total maximum transmission times. The worst total maximum transmission time, excluding the Ramp MCE, is 5035 μs and the total maxi-

id	arithmetic mean(μs)	average maximum(μs)	total maximum(μs)	jitter(μs)
0x1	544.44	1587.46	3432	3127
0x2	547.84	1641.14	3738	3433
0x3	552.38	1727.59	3585	3280
0x1c	665	2379.73	4195	3890
0x1d	670.97	2485.75	4653	4272
0x1e	677.98	2571	4424	4119
0x1f	911.59	3958.95	23115	22810

Table 6.18: Mean statistics of the transmission times of a test system with 4 CSDs and a low priority Ramp MCE

	CSDR	CSD0	CSD1	CSD2
MCE	0x1f	0x1c	0x17	0x1e
total maximum(μs)	23115	4195	5035	4424

Table 6.19: Total maximum transmission times of each CSD of a test system with 4 CSDs and a low priority Ramp MCE

mum transmission times of each CSD are similar. Furthermore, a higher MCE priority does not guarantee a low total maximum transmission time e.g. MCE 0x17 has the 7th highest priority.

Deviations of the Transmission Times

The measured variances of all MCEs are stable until the end of the test run. Figure 6.8 shows measured variances of the second lowest (MCE 0x1e) and highest (MCE 0x1) priority MCE for the test rounds 1000 to 1250 in μs. Furthermore, the x-axis shows the round number and the y-axis shows the measured variances of MCE 0x1 (node_0_1_variance) as well as MCE 0x1e (node_0_1e_variance). The measured variances of all MCEs are solid until test round 1180. At test round 1180 the measured variance increases for all MCEs but not for MCE 0x1. The increase for low priority MCEs is clearer than for high priority MCEs. This is shown by the measured variances of MCEs 0x1 and 0x1e in Figure 6.8. Furthermore, the measured variances of the Ramp MCE extremely raises at the end of the test round due to the full utilization of the communication medium and therefore a fully filled PSQ of the Ramp CSD. Moreover, the bandwidth consumption at test round 1180 is 287 kbit/s or 57.4% of 500 kbit/s.

Table 6.20 shows the deviation of the average transmission times (mean deviation), the deviation of the maximum transmission times (max deviation), the measured average variance of the transmission times (average var) and the deviation of the measured variances of the transmission times (var deviation) of the highest and lowest priority MCE of each CSD for the complete test run (the Ramp CSD has a single MCE).

The mean deviations hardly correlate with the priority of the MCEs and increase with the priority of the MCEs but not the Ramp MCE.

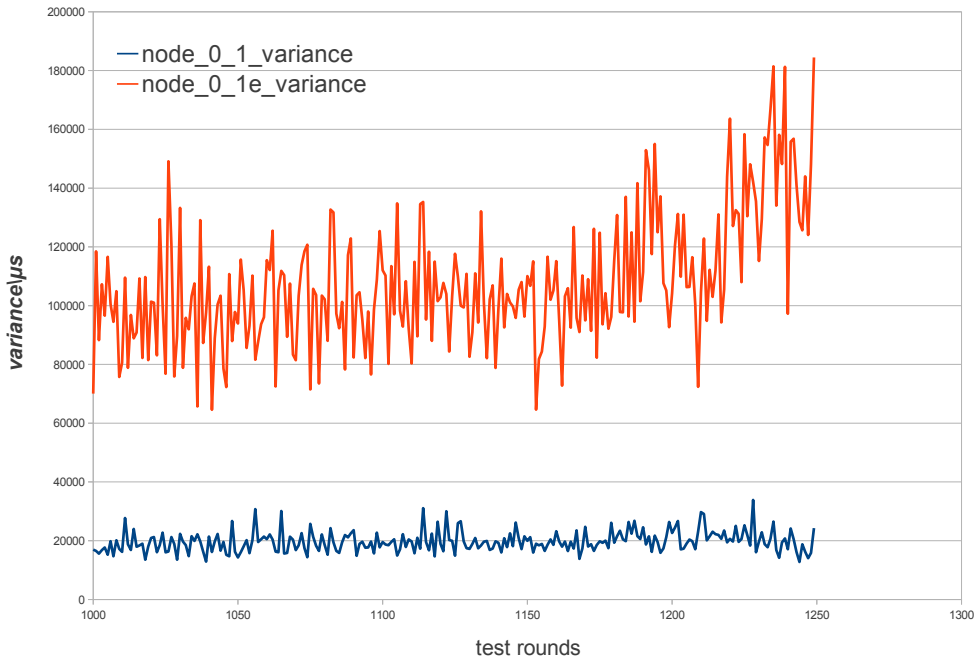


Figure 6.8: Measured variances of MCEs 0x1 and 0x1e in the test system with 4 CSDs and a low priority Ramp MCE

id	mean deviation(μs)	max deviation(μs)	average var(μs)	var deviation(μs)
0x1	17.15	324.04	17689.81	3169.77
0x2	17.65	343.13	19608.21	3646.06
0x3	18.66	369.87	21791.44	3967.8
0x1c	32.96	392.49	80622.77	14534.33
0x1d	34.3	419.43	87359.59	15886.31
0x1e	36.26	422.75	95346.38	18596.14
0x1f	2136.7	2567.8	195666.82	532328.66

Table 6.20: Variance statistics of the transmission times of a test system with 4 CSDs and a low priority Ramp MCE

The max deviations hardly correlate with the priority of the MCEs and linearly raise with the priority of the MCEs but not the Ramp MCE.

The measured average variances hardly correlate with the priority of the MCEs and increase extremely in special compared to the Ramp MCE.

The var deviations hardly correlate with the priority of the MCEs and increase strongly. Furthermore, the var deviation of the Ramp MCE is extremely increased when compared to the other var deviations.

id	mean(messages/round)	min	max	deviation(messages/round)
0x1	622.18	601	644	7.2
0x2	578.7	554	599	7.02
0x3	541.34	518	563	6.75
0x4	508.4	487	527	6.47
0x5	479.05	456	499	6.38
0x6	452.97	434	472	6.09
0x7	429.17	413	450	5.89
0x40	108.7	100	119	3.17
0x41	107.3	97	118	2.98
0x42	105.97	96	117	3.01
0x43	104.47	95	114	2.97
0x44	103.32	92	113	3.05
0x45	101.79	93	112	2.94
0x46	100.77	92	111	2.9
0x47	606.02	96	19248	1758.52

Table 6.21: Mean, minimum, maximum and standard deviation of the send behavior of a test system with 8 CSDs and with a low priority Ramp MCE

6.6 8 CSDs - Low Priority Ramp MCE

This Subsection presents the results obtained by the test run with 8 CSDs and with a low priority Ramp MCE.

Network Utilization

The maximum send capacity of the Ramp MCE is 1921 TMIs per second and the communication medium has a full utilization starting at test round 996. At test round 996 the bandwidth consumption is 456 kbit/s or 91.2% of 500 kbit/s. The send behaviour of the CSDs is stable (except for the Ramp MCE) during the complete test run and the Ramp MCE does not cut off the remaining MCE because the Ramp MCE has the lowest priority (0x47). Furthermore, the send omissions of the Ramp MCE 0x47 starts to raise at test round 996.

Table 6.21 shows the average send frequency of the TMIs (mean), the total minimum send frequency of the TMIs (min), the total maximum send frequency of the TMIs (max) and the mean deviation of the send frequencies of the TMIs (deviation) of the highest and lowest priority MCE of each CSD (the Ramp CSD 0 only has a single MCE) of the complete test run.

The Ramp MCE has a high mean as well as a high max because the emission of TMIs increases exponentially during the test rounds. Furthermore, as defined in Chapter 4 the send frequencies correlate with the priority of the MCEs.

Average Transmission Times

The total maximum and average transmission times of all MCEs are stable until the end of the test run. Figure 6.9 shows the average as well as total maximum transmission times of the second lowest (MCE 0x46) and highest (MCE 0x1) priority MCE for the test rounds 800 to 1000 in μs. Furthermore, the x-axis shows the round number. The y-axis shows the total maximum transmission times of MCE 0x1 (node_0_1_max) as well as of MCE 0x46 (node_0_46_max) and the average transmission times of MCE 0x1 (node_0_1_average) as well as of MCE 0x46 (node_0_46_average) in μs. The maximum transmission times of all MCEs are solid until test

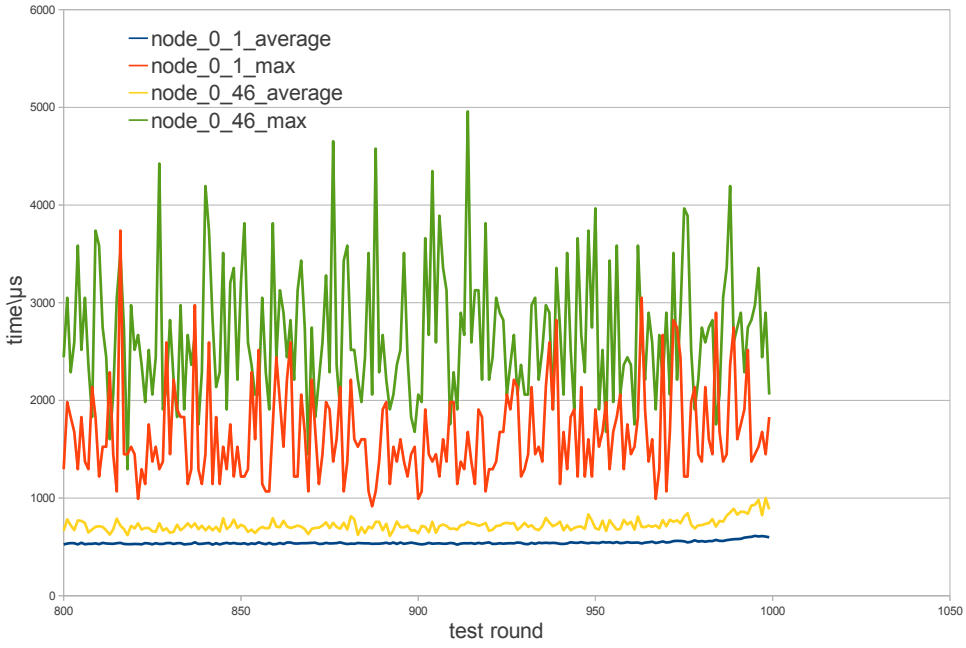


Figure 6.9: Total maximum and average transmission times of MCE 0x1 and MCE 0x46 in the test system with 8 CSDs and a low priority Ramp MCE

round 970. Furthermore, the average transmission times of the MCEs but not the Ramp MCE slightly increases at test round 970. This is shown by the average and total transmission times of MCEs 0x1 and 0x46 in Figure 6.7. Furthermore, the transmission times of the Ramp MCE raises extremely at the end of the test round due to the full utilization of the communication medium and therefore a fully filled PSQ of the Ramp CSD. Moreover, the bandwidth consumption at test round 970 is 247 kbit/s or 49.4% of 500 kbit/s.

Table 6.22 shows the average transmission time (arithmetic mean), the average maximum transmission time (average maximum), the total maximum transmission time (total maximum) and message transmission time jitter of each interval (jitter) of the highest and lowest priority

id	arithmetic mean(μs)	average maximum(μs)	total maximum(μs)	jitter(μs)
0x1	533.61	1601.5	4882	4577
0x2	536.33	1591.8	4958	4653
0x3	539.85	1632.01	3966	3661
0x4	540.66	1626.8	5950	5645
0x5	543.51	1638.12	4119	3814
0x6	545.95	1627.57	4577	4272
0x7	548.75	1664.42	5035	4730
0x40	686.9	2418.74	5187	4806
0x41	690.27	2426.14	4882	4501
0x42	688.92	2482.68	5569	5188
0x43	693.2	2506.19	6026	5645
0x44	696.33	2559.8	6560	6179
0x45	698.04	2596.88	5797	5416
0x46	700.49	2615.94	5721	5340
0x47	705.86	3464.72	26700	26395

Table 6.22: Mean and maximum statistics of the transmission times of a test system with 8 CSDs and with a low priority Ramp MCE

	CSDR	CSD0	CSD1	CSD2	CSD3	CSD4	CSD5	CSD6
MCE	0x47	0x40	0x3a	0x3b	0x43	0x44	0x1b	0x46
total maximum(μs)	26700	5187	5340	5950	6026	6560	5950	5721

Table 6.23: Total maximum transmission times of each CSD of a test system with 8 CSDs and a low priority Ramp MCE

MCE of each CSD for the complete test run (the Ramp CSD has a single MCE).

The arithmetic mean correlates with the priority of the MCEs and linearly increases with the MCE priority.

The average maximum correlates with the priority of the MCEs (especially for low priority MCEs) and apparently increases with the priority of the MCEs. The increase of the average maximum applies in especially only to the Ramp MCE.

The total maximum and therefore the jitter does not correlate with the priority of the MCEs. Furthermore, the total maximum and therefore jitter are extremely raised for the Ramp MCE due to the full utilization of the communication medium at the end of the test run.

Table 6.23 shows the total maximum transmission time for each CSD (CSDR denotes the Ramp MCE) and the MCEs which generated the total maximum transmission times. The worst total maximum transmission time, excluding the Ramp MCE, is 6560 μs and the total maximum transmission times of each CSD are similar. Furthermore, a higher MCE priority does not guarantee a low total maximum transmission time e.g. MCE 0x17 has the 4th highest priority of CSD 5.

Deviations of the Transmission Times

The measured variances of all MCEs are stable until the end of the test run. Figure 6.10 shows the measured variances of the second lowest (MCE 0x46) and highest (MCE 0x1) priority MCE for the test rounds 800 to 1000 in μs. Furthermore, the x-axis shows the round number and the y-axis shows the measured variances of MCE 0x1 (node_0_1_variance) as well as MCE 0x46 (node_0_46_variance) in μs. The measured variances of all MCEs but not the Ramp MCE are

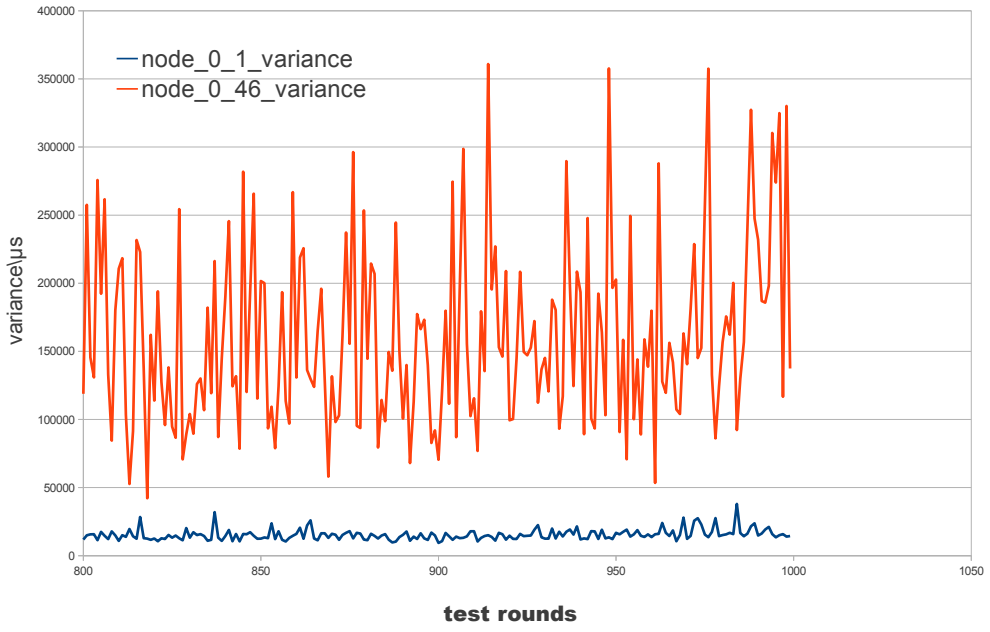


Figure 6.10: Measured variances of MCEs 0x1 and 0x46 in the test system with 4 CSDs and a low priority Ramp MCE

solid during the complete test run. At the very end of the test run the variances of the low priority MCEs but not the Ramp MCE increases very little. This is shown by the measured variances of MCEs 0x1 and 0x46 in Figure 6.9. Furthermore, the measured variances of the Ramp MCE raises extremely at the end of the test round due to the full utilization of the communication medium and therefore a fully filled PSQ of the Ramp CSD.

Table 6.24 shows the deviation of the average transmission times (mean deviation), the deviation of the maximum transmission times (max deviation), the measured average variance of the transmission times (average var) and the deviation of the measured variances of the transmission time (var deviation) of the highest and lowest priority MCE of each CSD for the complete test run (the Ramp CSD has a single MCE).

The mean deviations correlates to the priority of the MCEs and increases by the priority of

id	mean deviation(μs)	max deviation(μs)	average var(μs)	var deviation(μs)
0x1	9.51	473.63	14516.22	3732.78
0x2	10.19	470.27	15528.26	3953.54
0x3	10.45	496.52	17069.29	4368.74
0x4	10.78	492.71	18175.62	4833.24
0x5	11.38	478.64	19530.64	5069.18
0x6	11.55	450.26	20654.55	5284.34
0x7	11.93	476.16	22027.4	5641.11
0x40	39.76	585.97	120796.67	46707.53
0x41	41.27	571.54	125059.95	49514.9
0x42	39.25	607.78	129242.52	48746.21
0x43	42.51	624.43	134139.46	54390.44
0x44	43.19	663.45	140913.87	57442.46
0x45	42.3	673.15	144646.08	59640.54
0x46	44.23	667.26	146886.37	59663.16
0x47	1052.02	1702.48	192958.27	320803.63

Table 6.24: Variance statistics of the transmission times of a test system with 8 CSDs and a low priority Ramp MCE

the MCEs. The Ramp MCE has extremely raised mean deviation.

The max deviations correlate with the priority of the MCEs and linearly raise for all MCEs but not the Ramp MCE. The Ramp MCE has extremely raised max deviation.

The measured average variances hardly correlates with the priority of the MCEs and increases extremely, in special for the Ramp MCE.

The var deviation hardly correlates with the priority of the MCEs and increases extremely. Furthermore, the var deviation of the MCEs extremely increases compared to the var deviation of the Ramp MCEs.

CHAPTER 7

Discussion

This Chapter discusses the results and the properties of the prototype.

7.1 Test System

This Subsection discusses the properties of the test setup.

Utilization of the Processors

The processors (CSDs) of the system have a high utilization in the test runs. Furthermore, the granularity of the local time base that was 100 μs seems to be the maximum possible trigger interval. A higher granularity yields to delays. During an interrupt interval that was 100 μs, 10,000 CPU cycles complete and during this time period a single run of the main loop must complete. A shorter time period seems to interrupt a single loop run (the calculations of the send times during the interrupt also have to complete during the 10,000 cycles). Therefore it is possible that the high granularity of the local time base affect the measurement of the transmission times.

Inaccuracy of the Global Time Base

The time stamps from the global time base show an unexpected behaviour. For example, the minimum transmission time of all MCEs in the test runs without Ramp MCE have a duration of 305 μs, 381 μs or 457 μs. Furthermore, the total maximum transmission time of MCE 0x1 in all test runs is either 991 μs or 1068 μs. It seems that the global time base has a granularity of 76 μs but all measured transmission times are calculated with factor of 0.0745 (1 tick correspond to 0.0745 microseconds). However this inaccuracy can be neglected as a CAN frame takes at least 281 μs for transmission (data Frame including bit stuffing - see Subsection 7.5) and the time difference of 24 μs corresponds to 8.54% (under 10%). Thus the minimum transmission times also have an adequate accuracy because of the an inaccuracy under 10%. Furthermore, the order

of the transmission times per message id is the main focus and therefore the average transmission times, maximum transmission times and measured variance are acceptably accurate.

Data Acquisition

There are more data sets at low bandwidth utilization as the bandwidth of the Ramp MCE increases exponentially (see Figure 6.1). However there are data sets with many values for a bandwidth consumption of up to 230 kbit/s. For example, in the test run "8 CSDs - High Priority Ramp MCE" there are in total 8000 test rounds and 230 test rounds are in the bandwidth cap between 225 kbit/s and 230 kbit/s. In this bandwidth cap about 4 Million TMIs were send in total. Furthermore, in the bandwidth cap between 230 - 300 kbit/s 21 Million TMIs were send in total. However CAN shows an adequate transmission behaviour for speeds up to 230 kbit/s and therefore higher bandwidth intervals are less interesting.

SMS

In the test setup, each message (TMI) is stored in the SMS until it is transmitted. Furthermore, in real systems such messages are replaced by messages of higher priority that can be triggered to be compatible with the CAN standard. Moreover, there were problems by the implementation of such routine. It was possible to remove the pending message but not to detect it if the message was successfully send or not (there is a small time cap between reading the status register and writing the command to remove the pending message). The check for the 'Tra Buffer empty' interrupt flag did not work properly. However on average the transmission times are less affected and the worst total maximum transmission time in each test run has a high informative value even if the total maximum transmission time can not be related to a special message id.

Transmission Errors

There are probably many transmission errors due to the settings of the bit rate that was set to 500 kbit/s and the usage of single unshielded wires that are used as an interconnection (CAN Bus) between the subscribers (CSDs). The ISO 11898 requires a twisted pair for a network speed of more than 125 kbit/s. Thus there are probably many transmission errors even if the wire lengths are short and therefore the transmission times are increased.

Total Maximum Send Time of MCE 0x1

The total maximum send time of MCE 0x1 is in all test runs limited to 1068 μs or 991 μs. It seems that there are constraints that yields to an accumulation of circumstances that delay the transport of TMI 0x1 to the maximum limit. There is probably a combination of errors including error frame emission, retransmissions and TMIs with lower priority that triggers at the Interframe Space position 3 or Overload Frames. Furthermore, the error counters are checked for determination of the Error Active state of the CSD and Overload Frames are not detectable. Thus it is not possible to determine the exact reason for this behaviour. This behaviour should be analyzed in future works.

7.2 Interpretation of Data

This Subsection provides an analysis of the obtained results.

4 CSDs and 8 CSDs - No Ramp MCE

The test setups "4 CSDs - No Ramp MCE" and "8 CSDs - No Ramp MCE" have the same utilization of 40% but in general, in the test setup with 8 CSDs, the transmission times are increased and vary much more.

The total maximum transmission times are extensively increased in the test setup with 8 CSDs. For example, the worst total maximum transmission time of the test system with 8 CSDs is increased by 56% when compared to the test system consisting of 4 CSDs.

Furthermore, the correlation of the total maximum transmission times with the priority of the messages is influenced by the number of messages. There is a clearer correlation of the prioritizes of the MCEs with the total maximum transmission times in the test setup with 4 CSDs.

The average transmission times as well as the average maximum transmission times of the test system consisting of 4 CSDs are slightly smaller when compared to the test system with 8 CSDs. Furthermore, the measured average variance of the transmission times is smaller in the test run consisting of 4 CSDs. For example, the measured average variance of the lowest priority MCE in the test setup with 8 CSDs is increased by 58.1% when compared to the lowest priority MCE in the test setup with 4 CSDs.

4 CSDs - High Priority Ramp MCE

The test setup "4 CSDs - High Priority Ramp MCE" has a stable transmission time behaviour if the utilization does not exceed 230 kbit/s (see Figure 6.3 and Figure 6.4). There are similar statistics for the transmission times in Table 6.10 (4 CSDs - High Priority Ramp MCE) and Table 6.2 (4 CSDs - No Ramp MCE). All statistical values are similar but not for the total maximum transmission times. For example, the worst total maximum transmission time of the test system with Ramp MCE is increased by 19.3% when compared to the test system without Ramp MCE. In the test setup with Ramp MCE there is a weaker correlation of the total maximum transmission times with the priorities of the MCEs.

The variances of both test setups are also similar as shown in Table 6.12 (4 CSDs - High Priority Ramp MCE) and Table 6.4 (4 CSDs - No Ramp MCE). This shows that the transmission fulfills in a similar way e.g. the collisions at the bus.

The test setup "4 CSDs - No Ramp MCE" has a volatile transmission time behaviour if the utilization is between 230 and 300 kbit/s. The average transmission times and the average maximum transmission times are increased in the test setup with Ramp MCE when compared to the test setup without Ramp MCE. For example, the average transmission time of MCE 0x1f in test system with Ramp MCE is increased by 17.8% when compared to the test system without Ramp MCE and the average total maximum transmission time of MCE 0x1f in test system

with Ramp MCE is increased by 30.8% when compared to the test system without Ramp MCE. Though the total maximum transmission times are significant increased. For example, the worst total maximum transmission time of the test system with Ramp MCE is increased by 38.6% when compared to the test system without Ramp MCE.

The deviation of the average transmission times is increased for the test setup with Ramp MCE. For example, the deviation of the average transmission time of MCE 0x1f of the test system with Ramp MCE is increased by 207.0% when compared to the test system without Ramp MCE. The deviation of the total maximum transmission times are increased. For example, the deviation of the total maximum transmission time of MCE 0x1f of the test system with Ramp MCE is increased by 56.0% when compared to the test system without Ramp MCE. Furthermore, the measured average variances and the deviations of the measured average variances are extremely increased. For example, the measured average variance (deviation of the measured average variances) of MCE 0x1f of the test system with Ramp MCE is increased by 104.8% (217%) when compared to the test system without Ramp MCE. This shows that there are many collisions at the emission of messages.

The test setup "4 CSDs - No Ramp MCE" has an odd transmission time behaviour if the utilization is between 300 and 350 kbit/s. This can be seen in Figure 6.3 as explained in Subsection 6.3 as well as in Figure 6.4 as detailed in Subsection 6.3.

8 CSDs - High Priority Ramp MCE

The test setup "8 CSDs - High Priority Ramp MCE" has a stable transmission time behaviour if the utilization does not exceed 230 kbit/s (see Figure 6.5 and Figure 6.6). There are similar statistics for the transmission times in Table 6.14 (8 CSDs - High Priority Ramp MCE) and Table 6.6 (8 CSDs - No Ramp MCE). Nearly all statistical values are similar except for the total maximum transmission times. The total maximum transmission times are increased for the test setup with Ramp MCE. For example, the worst total maximum transmission time of the test system with Ramp MCE is increased by 20.2% when compared to the test system without Ramp MCE. In the test setup with Ramp MCE there is a weaker correlation of the total maximum transmission times with the priorities of the MCEs.

The variances of both test setups are also similar as shown in Table 6.16 (8 CSDs - High Priority Ramp MCE) and Table 6.8 (8 CSDs - No Ramp MCE). This shows that the transmission fulfills in a similar way e.g. the collisions on the bus.

The test setup "8 CSDs with Ramp MCE" has an unacceptable transmission time behaviour if the utilization is between 230 and 300 kbit/s. The average transmission times are increased in the test setup with Ramp MCE when compared to the test setup without Ramp MCE. For example, the average transmission time of MCE 0x47 in test system with Ramp MCE is increased by 20.3% when compared to the test system without Ramp MCE. Furthermore, the average maximum transmission times are increased. For example, the average total maximum transmission time of MCE 0x47 in the test system with Ramp MCE is increased by 36.9% when compared to the test system without Ramp MCE. Moreover, the worst maximum transmission time is

increased by 47.2% in the test setup with a high priority Ramp MCE.

The deviation of the average transmission times (deviation of the total maximum transmission time) is highly increased for the test setup with Ramp MCE. For example, the deviation of the average transmission time (deviation of the total maximum transmission time) of MCE 0x47 in the test setup with Ramp MCE is increased by 152.2% (61.3%) when compared to the test setup without Ramp MCE.

Furthermore, the measured average variances of the test round are highly increased. For example, the measured average variance (deviation of the measured average variances) of MCE 0x47 of the test system with Ramp MCE is increased by 128.6% (212.8%) when compared to the test system without Ramp MCE. This shows that there are many collisions at the emission of messages.

The test setup "8 CSDs - No Ramp MCE" has an odd transmission time behaviour if the utilization is between 230 and 300 kbit/s. This can be seen in Figure 6.5 as explained in Subsection 6.4 as well as in Figure 6.6 as detailed in Subsection 6.4.

4 CSDs - Low Priority Ramp MCE

The test setups "4 CSDs - Low Priority Ramp MCE" has a basic utilization of 40%. Furthermore, the lowest priority MCE (MCE 0x1f) is the Ramp MCE. Therefore the Ramp MCE disturbs the communication of the CSDs and thus the transmission times of the Ramp MCE are excluded from the analysis.

The average transmission times as well as the average total maximum transmission times are similar when compared to the test setup 4 CSDs - No Ramp MCE. However at the end of the test run the transmission times slightly increase as shown in Figure 6.7. Furthermore, the total maximum transmission times are increased. For example, the worst total maximum transmission time in test system with Ramp MCE is increased by 15.8% when compared to the test system without Ramp MCE.

The deviations are also similar except for the deviations of the average transmission times when compared to the test setup without Ramp MCE. The deviation of the average transmission times are significantly increased. For example, the deviation of the average transmission time of MCE 0x02 in the test setup with Ramp MCE is increased by 211.3% when compared to the test setup without Ramp MCE. Furthermore, the measured average variances of the transmissions times are increased starting at a utilization of 57% for low priority MCEs (see Figure 6.8).

8 CSDs - Low Priority Ramp MCE

The test setups "8 CSDs - Low Priority Ramp MCE" has a basic utilization of 40%. Furthermore, the lowest priority MCE (MCE 0x47) is the Ramp MCE. Therefore the Ramp MCE disturbs the communication of the CSDs and thus the transmission times of the Ramp MCE are excluded from the analysis.

The average transmission times, average total maximum transmission times and total maximum transmission times are similar when compared to the test setup 8 CSDs - No Ramp MCE. However at the end of the test run the average transmission times slightly increase as shown in Figure 6.7.

The deviations and the average variances are also similar but not for the deviations of the average transmission times to the test setup without Ramp MCE. The deviation of the average transmission times are significantly increased for high priority MCEs. For example, the deviation of the average transmission time of MCE 0x02 in the test setup with Ramp MCE is increased by 94.8% when compared to the test setup without Ramp MCE.

7.3 Collision Probability

A collision happens if two messages are triggered for sending in a time interval that is shorter than required to transmit all triggered messages. Furthermore, collided messages can be seen as a single message emission and further collisions extend the emission time of such a merged message. Figure 7.1 shows collisions. Assume a system with 3 CSDs (subscribers) and each

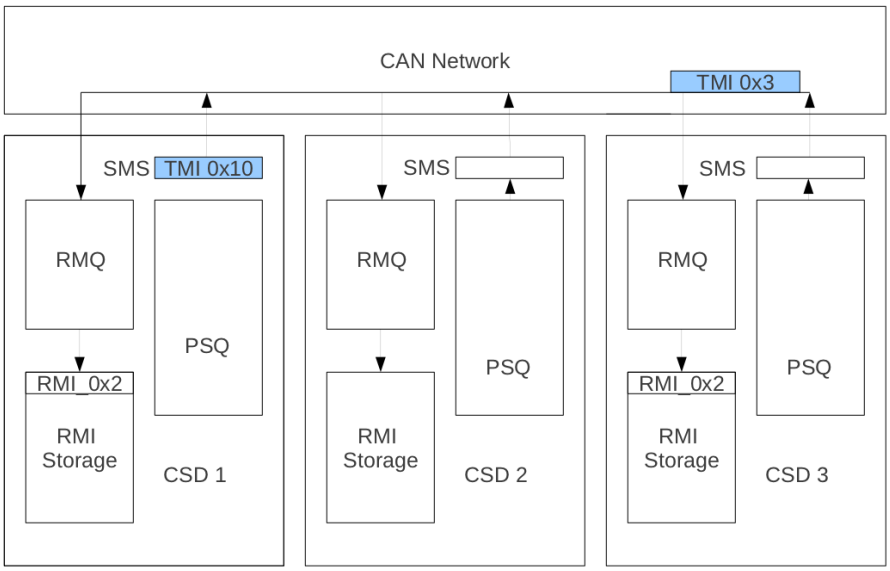


Figure 7.1: Blocking of a triggered messages

CSD has an empty PSQ and empty SMS. In addition, the CAN bus is not in use for any transmission. Furthermore, TMI 0x2 triggers at CSD 2, is pushed into the SMS and emits. After the start of the emission TMI 0x10 triggers at CSD 1 and is pushed into the SMS. Furthermore, TMI 0x3 triggers at CSD 3 and is pushed into the SMS. TMIs 0x3 and 0x10 can not be sent if the bus is occupied by the emission of TMI 0x2. Furthermore, the transmission of TMI 0x2 completes and therefore TMI 0x2 is converted into RMI 0x2. RMI 0x2 is stored in the RMI Storages of CSD 1 and CSD 3. At the completion of TMI 0x2, TMI 0x3 starts to emit. Figure 7.1 shows a snapshot of this time point. Using this constellation TMI 0x10 will be emitted at the end and all

RMIs except RMI 0x2 will have an increased transmission time. Furthermore, if TMI 0x2 has not been triggered, than TMI 0x10 would be emitted instantly without a queuing penalty.

There exits more complicated szenarios that are possible and therefore, depending on the scenario, the transmission times fluctuate. This example shows how a collision influences the average transmission times, total maximum transmission times and the dispersion of transmission times. The collision probability is higher if more messages can trigger during an interval. As a consequence, higher collision probability leads to more collisions during the communication of CAN subscribers (CSDs).

The increased collision probability and therefore timing behaviour is shown in the test runs without Ramp MCE (see Subsection 7.2). Even if the utilization is the same, the number of CAN messages (MCEs) influences the message timing behaviour especially for low priority CAN messages. Note that the emission rate is limited for each CAN message (MCE) and that the total maximum transmission time is smaller than any MINT. Thus the number of the CAN messages is an important dimension.

7.4 Errors and Overload Frames

In CAN errors influence the time required for the transmissions of messages (TMIs) because each data frame that is rejected is always followed by an error frame and thus has to be retransmitted. This may decrease the time required for transmission of a single prioritized message that is waiting for transport but in contrast the time required for transmission of other messages increase because of the delay at the communication medium by the erroneous frame. Furthermore, each erroneous frame also increase the utilization of the communication medium and so the collision probability of additional frames. Such delay depends on the frame length (number of bits) and the bit time. This elongation is caused by the transmission of the erroneous frame until a receiver or sender will interfere and this interference causes an error frame consisting 14 - 20 bit times (see Subsection 3.1), followed by an Interframe Space of a 3 bit time size (see Subsection 3.1).

This delay will also increase the chance of additional collisions which in turn lead to longer transmission times because message transmissions are temporarily blocked. A single CAN bus can only transport a single frame that consists of a single message.

For example, assume multiple subscribers that are attached to a single CAN bus and a data frame that is in emission (frame with identifier 2). Furthermore, a data frame is ready for transmission at a different subscriber (frame with identifiers 3). In this case the emission of frame 2 will be rejected by an error frame. During this time another frame with identifier 1 gets ready for transmission and is emitted after the previous error frame. In analogy, after frame 1 had been transmitted frame 2 will be retransmitted and frame 3 transmitted (see Figure 7.2, IS stands for Interframe Space). Therefore the transfer of frame 1 will be completed earlier because the erroneous frame 2 was rejected and so frame 1 was send off earlier. As a consequence frame 3 will be processed later due to the unsuccessful transmission. In addition, frame 1 will also increase the delay for frame 2. This principle generally applies to all messages that are waiting for transmission at their corresponding subscribers that forward frame 2 and 3 and may be extended

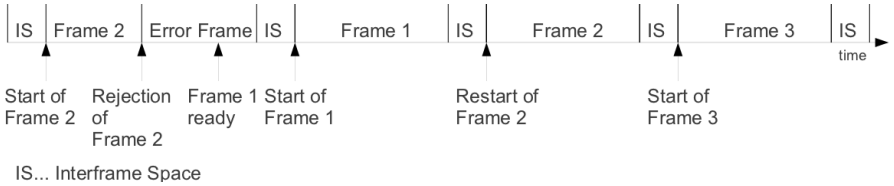


Figure 7.2: Delay of frames due to an erroneous frame

by triggering additional frames, etc.

Messages can also be delayed by overload frames. Overload frames are generated after the transmission of a frame by overburdened CAN Controllers to block the traffic at the communication medium so that the corresponding CAN Controller has time to complete (see Subsection 3.1).

7.5 Emission Rate and Utilization

The utilization of the communication medium vitally influences the timing behaviour of messages. The transmission time behaviour is stable up to a bandwidth of 230 kbit/s that corresponds to an utilization of 46% (see Subsection 7.2 and Subsection 7.2). Furthermore, the bandwidth consumption is calculated without bit stuffing. For example, the data frame consisting of stuff bits for TMI 0x1 was made up as follows (see Figure 3.3): 2 stuff bits for SOF and Identifier A (identifier is 0x1), 3 stuff bits for Identifier B (identifier is 0x1) and 7 stuff bits for the DLC, the Data Field and CRC Checksum by assuming 35% of the worst-case stuff bits (every 4th bit is a stuff bit, see Subsection 2.3). Therefore a transmission frame consists of 143 bits and thus by assuming a frame length of 143 bits the maximum utilization is 50%. However this limit is an optimistic limit. Thus the bandwidth consumption should be chosen less optimistically. By assuming a penalty of 20% the maximum utilization of a CAN bus is 40%.

A full utilization does not restrict the transmission of messages (TMIs) at all. For example, as explained in Subsection 6.4 all messages are transported up to a bandwidth consumption (assumed frame lengh is 131 bits) of 450 kbit/s (450 kbit/s is the maximum utilization). The transport of all triggered messages, even at a full but not overloaded utilization, can be seen in Figure 6.1 and Figure 6.2 (test run consisting of 800 test rounds, 4 CSDs - High Priority Ramp MCE). The disadvantage of a high utilization is the increased transport time of messages. A higher emission rate of TMIs leads to a higher possibility for collisions and thus to higher average transmission times. Collisions especially influence the timing behaviour of low priority MCEs due to CSMA/CR (see Section 3.1). Furthermore, collisions also increase the average transmission times of high priority messages as a transmission can not be aborted (see Subsection 7.2 and Subsection 7.2). This can be seen in Figure 6.7 and Figure 6.8. When a collision occurs, all transmission times are increased for the following TMIs that trigger during the duration of the collision. In addition, the transmission times persist if many messages (TMIs) trigger at the same time i.e. during an interval that is too short to schedule all transmissions without

collision.

A higher emission rate also increases the total maximum transmission time and so the average transmission times in a CAN network. There are more collision possibilities and thus messages are blocked for a longer time. If there is a collision the chance is increased that there is another collision. For example, assume a data frame that is in emission. Furthermore, a message (TMI 0x3) triggers for sending at the start of the frame that is in emission but can not be transported when the communication medium is occupied. After that a message (TMI 0x2) triggers at another subscriber at the end of the frame that is in transport and this message can also not be transported. Therefore TMI 0x3 has a three times higher transport time than possible for the best case (instantly transported). Such examples can be enhanced. The limitations of such total maximum transmission times is the limitation of the emission rate of each message and the number of messages. These two parameters determine the utilization of the CAN bus.

The fluctuation of the transmission times is also influenced by the collision probability. The collision probability is the parameter for the dispersion of the transmission times. A high collision probability yields to a high variance of the transmission times. Furthermore, the emission rate per message id and the number of messages are the main properties that influence the variances of the transmission times of messages. Moreover, the priority determines the parameter of the variance. See Subsection 7.2 for the interpretation of data and Chapter 6 for details.

7.6 Future Work

The test system can be extended by a LAN interface to connect the CEU to a host computer. The LAN connection can be used to transmit the received RMIs to the host computer upon receipt from the CEU and therefore the statistical data can be calculated at the host computer. The data evaluation at a host computer is more selective.

The distribution and number of messages can be varied in future test runs. Therefore, test runs without RAM MCE with different utilizations can be generated to obtain more statistical data. Furthermore, the test system can be combined with a fault injection system so that error/overload frames are emitted during a test round to obtain statistical data under erroneous conditions. In addition, retransmission requests can be considered in the simulation model. Therefore a CSD requests a message from another CSD and therefore the total end-to-end transmission time is measured i.e. the difference between receipt of the requested message and the generation of the request message. Furthermore, in a worst-case response time analysis (see Subsection 2.3) it is assumed that all emitted messages collide most. Therefore it is assumed that all messages are triggered at the start simultaneously and emitted during the shortest possible interval (MINT only). This is a very pessimistic model that is not well suited for the CAN standard as CAN messages are not used for reliable communication. Moreover, in a CAN network each subscriber is responsible for his error management and therefore error frames will lead to a violation of assumptions made by the worst-case response time analysis. In addition, the minimum send periods are not guaranteed. However the statistical data achieved from the

prototype can be used to derive a stochastic model that guarantees an in time message transmission with a certain probability (distribution). Furthermore, such a stochastic model can be used in combination with the CAN Router of the Vienna University of Technology (see Subsection 3.2) to determine the minimum time of messages with the same identifier because the Router has a global view of the communication.

CAN stars can be evaluated but the test system has to be adopted if the emitted messages are not broadcast to all CSDs. Furthermore, in the current implementation each received RMI is checked at the CEU for a legal identifier so that the identifier belongs to a MCE. Therefore there exist RMIs that own an identifier from a MCE but are not generated from the MCE that relates to them. In a CAN bus (or CAN Router that broadcasts to all subnetworks but not the emitting subnetwork) such errors may be detected by the receive omissions. However such an error can be checked by an emission log (log of each created TMI) of each MCE and by a comparison with the received RMIs.

7.7 Conclusion

CAN has a good transmission time behaviour if the utilization does not exceed 40%. The average transmission times and the variance of the transmission times are limited and especially for high priority messages but the total maximum transmission times vary considerably. Thus CAN is very well suited for unreliable systems or systems that have an independent backup system.

List of Acronyms

BSTEP Bandwidth Step
CAN Controller Area Network
CEU Central Evaluation Unit
CIS CAN Interface Subsystem
CPN Colored Petri Nets
CSD CAN Simulation device
CSMA/CR Carrier Sense Multiple Access/Collision Resolution
DAG Directed Acyclic Graph
DLL Data Link Layer
ECU Electronic Control Unit
FIFO First In - First Out
FPGA Field-Programmable Gate Array
gcd greatest common divisor
HIL Hardware in the Loop
HSMC High-Speed Mezzanine Cards
id identifier
I/O Input/Output
IP Intellectual Property
IRQ Interrupt Request
ISO International Organization for Standardization
lcm least common multiple
LLC Logical Link Control
MAC Medium Access Control
MATLAB Matrix Laboratory
MCE Message Creation Entity
MDI Medium Dependent Interface
MINT Minimum Interarrival Time
MPSoC Multiprocessor System-on-Chip
MRT Message Reception Time
MTI Message Time Interval
NoC Network-on-a-Chip
NRZ Non Return to Zero
OSI Open System Interconnection
PMA Physical Medium Attachment

pmf probability mass function
PSQ Prioritized Send Queue
RAND Random Send Time
RMA Resource Management Authority
RMI Received Message Instance
RMINT Ramp Minimum Interarrival Time
RMQ Receive Message Queue
RSTEP Round Bandwidth Step
SMINT Start Minimum Interarrival Time
SoB System on Board
SoC System on Chip
SOPC System on a Programmable Chip
TBAND Total Bandwidth Consumption
TDMA Time Division Multiplex Access
TISS Trusted Interface Subsystem
TMI Triggered Message Instance
TNA Trusted Network Authority
TTM Trigger Time of a Message
TTNoC Time-Triggered Network-on-Chip
TTSoC Time-Triggered System-on-Chip
TTT Total Transmission Time

Bibliography

[AHM05] Parosh Aziz Abdulla, Noomene Ben Henda, and Richard Mayr. Verifying Infinite Markov Chains with a Finite Attractor or the Global Coarseness Property, 2005.

[Alt08] Altera, 101 Innovation Drive, San Jose, CA 95134. *Stratix III Development Kit Reference Manual*, 1.2 edition, 2008.

[Alt09] Altera, 101 Innovation Drive, San Jose, CA 95134. *Nios II Software Developer's Handbook*, 2009.

[Alt11a] Altera, 101 Innovation Drive, San Jose, CA 95134, USA. *Avalon Interface Specifications*, 11.0 edition, May 2011.

[Alt11b] Altera, 101 Innovation Drive, San Jose, CA 95134. *Embedded Design Handbook*, 2011. Description of the ALTERA embedded developing tools.

[Alt12] Altera, 101 Innovation Drive, San Jose, CA 95134. *Quartus II Handbook*, 12.0 edition, 2012. Description of the ALTERA Quartus II design software.

[Bac05] M. Bacic. On hardware-in-the-loop simulation. In *Decision and Control, 2005 and 2005 European Control Conference. CDC-ECC '05. 44th IEEE Conference on*, pages 3194 – 3198, dec. 2005.

[BMP07] Marko Bago, Sinisa Marijan, and Nedjeljko Peric. Modeling Controller Area Network Communication, 2007.

[BPA09] M. Barranco, J. Proenza, and L. Almeida. Boosting the Robustness of Controller Area Networks: CANcentrate and ReCANcentrate. *Computer*, 42(5):66 –73, may 2009.

[BRNPA04] Manuel Barranco, Guillermo Rodriguez-Navas, Julián Proenza, and Luís Almeida. CANcentrate: An active star topology for CAN networks, 2004.

[BvL11] H. Broeders and R. van Leuken. Extracting behavior and dynamically generated hierarchy from systemc models. In *Design Automation Conference (DAC), 2011 48th ACM/EDAC/IEEE*, pages 357 –362, june 2011.

[DBBL] Robert I. Davis, Alan Burns, Reinder J. Bril, and Johan J. Lukkien. Controller Area Network (CAN) Schedulability Analysis: Refuted, Revisited and Revised.

[fS96] International Organization for Standardization. Information technology - Open Systems Interconnection - Basic Reference Model: The Basic Model, 1996. Reference number: ISO/IEC 7498-1:1994(E).

[fS03] International Organization for Standardization. Road vehicles – Controller area network (CAN) – Part 1: Data link layer and physical signalling, 2003. Reference number: ISO 11898-1:2003.

[FYS10] D. Fennibay, A. Yurdakul, and A. Sen. Introducing hardware-in-loop concept to the hardware/software co-design of real-time embedded systems. In *Computer and Information Technology (CIT), 2010 IEEE 10th International Conference on*, pages 1902 –1909, 29 2010-july 1 2010.

[GLMS02] T. Groetker, S. Liao, G. Martin, and S. Swan. *System design with SystemC*. Kluwer Academic Publishers, 2002.

[HC10] Luhe Hong and Jianli Cai. The Application Guide of Mixed Programming between MATLAB and Other Programming Languages, 2010.

[HWC04] Jörg Henkel, Wayne Wolf, and Srimat Chakradhart. On-chip networks: A scalable, communication-centric embedded system design paradigm, 2004.

[HWY+11] Chun-Ming Huang, Chien-Ming Wu, Chih-Chyau Yang, Shih-Lun Chen, Chi-Shi Chen, Jiann-Jenn Wang, Kuen-Jong Lee, and Chin-Long Wey. Programmable System-on-Chip for Silicon Prototyping. Technical Report 3, IEEE Transactions on Industrial Elektronics, March 2011.

[Ing09] Ingenieurbüro für IC-Technologie, Kleiner Weg 3, 97877 Wertheim, Germany. *IFI NIOSII CAN Module*, 2009.5 rev 9.0 edition, May 2009.

[JKW07] Kurt Jensen, Lars Michael Kristensen, and Lisa Wells. *Coloured Petri Nets and CPN Tools for Modelling and Validation of Concurrent Systems*. Springer-Verlag, 2007.

[KB02] Michael Keating and Pierre Bricaud. *Reuse Methodology Manual for System-on-a-Chip Designs*. Kluwer Academic Publishers, 3 edition, 2002.

[KOF12] Roland Kammerer, Roman Obermaisser, and Bernhard Frömel. A Router for the Containment of Timing and Value Failures in CAN. *EURASIP Journal on Embedded Systems*, 2012. note: accepted, to be published in EURASIP Journal on Embedded Systems.

[Kop97] Hermann Kopetz. *Design Principles for Distributed Embedded Applications*. Kluwer Academic Publishers, 7 edition, 1997.

[Kop08] Hermann Kopetz. The Complexity Challenge in Embedded System Design. In *11th IEEE Internation Symposium on Object/component/service-oriented Real-time distributed Computing(ISORC)*, pages 3 – 12, Orlando, Florida, USA, 2008.

[LPL11] Bin Ling, Fengchao Peng, and Ailan Li. The car body control bus design based on CAN/LIN bus, 2011.

[LWFM07] Bin Lu, Xin Wu, H. Figueroa, and A. Monti. A Low-Cost Real-Time Hardware-in-the-Loop Testing Approach of Power Electronics Controls. *Industrial Electronics, IEEE Transactions on*, 54(2):919 –931, april 2007.

[Mat] MathWork. SIMULINK - Simulation and Mopdel-Based Design. `http://www.mathworks.com/products/simulink/index.html`. Accessed: 9/05/2012.

[Mic03] Microchip, 2355 West Chandler Blvd., Chandler, AZ 85224-6199. *MCP2551 - High-Speed CAN Transceiver*, ds21667d edition, 2003.

[MMTS11] Saad Mubeen, Jukka Mäki-Turja, and Mikael Sjödin. Extending Response-Time Analysis of Controller Area Network (CAN) with FIFO Queues for Mixed Messages, 2011.

[Mol06] E. Mollick. Establishing Moore's Law. *Annals of the History of Computing, IEEE*, 28(3):62 –75, july-sept. 2006.

[NHN03] Thomas Nolte, Hans Hansson, and Christer Norström. Probabilistic Worst-Case Response-Time Analysis for the Controller Area Network, 2003.

[Nov09] Jiri Novak. Flexible approach to the controller area networks test and evaluation. In *IEEE International Workshop on Intelligent Data Acquisition and Advanced Computing Systems: Technology and Applications*, September 2009.

[OAFAA06] H. F. Othman, Y. R. Aji, F. T. Fakhreddin, and A. R. Al-Ali. Controller Area Networks: Evolution and Applications. Technical report, Computer Engineering Department American University of Sharjah, UAE, 2006. in IEEE data base.

[OMFK08] B. Osterloh, H. Michalik, B. Fiethe, and K. Kotarowski. SoCWire: A Network-on-Chip approach for reconfigurable System-on-Chip Designs in Space Applications, 2008.

[Pau08] Christian Paukovits. *The Time-Triggered System-on-Chip Architecture*. PhD thesis, Vienna University of Technology, 2008.

[PV03] Luís Miguel Pinho and Francisco Vasques. Timing Analysis of Reliable Real-Time Communication in CAN Networks, 2003.

[Ric02] Pat Richards. *A CAN Physical Layer Discussion*. Microchip Technology Inc., 2002.

[Rob91] Robert Bosch GmbH, Postfach 30 02 40, D-70442 Stuttgart, Germany. *CAN Specification Version 2.0*, March 1997 edition, September 1991.

[SBR+07] Benaoumeur Senouci, Aimen Bouchhima, Frédéric Rousseau, , and Frédéric Pétrot. Prototyping Multiprocessor System-on-Chip Applications: A Platform-Based Approach. Technical Report 8, IEEE distributed systems, May 2007.

[SWM+06] Resve Saleh, Steve Wilton, Shahriar Mirabbasi, Alan Hu, Mark Greenstreet, Guy Lemieux, Partha Pratim Pande, Cristian Grecu, and Andre Ivanov. System-on-Chip: Reuse and Integration, June 2006.

[TBW95] K. Tindell, A. Burns, and A J . Wellings. CALCULATING CONTROLLER AREA NETWORK MESSAGE RESPONSE TIMES. *Control Eng. Practice*, 3(8), 1995.

[WWM08] Ahmed Amine Jerraya Wayne Wolf and Grant Martin. Multiprocessor System-on-Chip (MPSoC) Technology, 2008.

[Zel11] Holger Zeltwanger. Die Zukunft von CAN und CANopen. `http://www.anybus.de/technologie/canopen.shtml`, July 2011. Accessed: 19/02/2012.

[ZNGSV09] Haibo Zeng, Marco Di Natale, Paolo Giusto, and Alberto Sangiovanni-Vincentelli. Stochastic Analysis of CAN-Based Real-Time Automotive Systems, 2009.