Bachelorarbeit

Arne Saggau

Sicherheitsorientierte Weiterentwicklung der mikrocontrollerbasierten Telemetrie in einem Formula Student Rennwagen durch Migration von TTCAN zu FlexRay

Fakultät Technik und Informatik
Department Informatik

Faculty of Engineering and Computer Science
Department of Computer Science

Arne Saggau

Sicherheitsorientierte Weiterentwicklung der mikrocontrollerbasierten Telemetrie in einem Formula Student Rennwagen durch Migration von TTCAN zu FlexRay

Bachelorarbeit eingereicht im Rahmen der Bachelorprüfung
im Studiengang Technische Informatik
am Department Informatik
der Fakultät Technik und Informatik
der Hochschule für Angewandte Wissenschaften Hamburg

Betreuender Prüfer : Prof. Dr. Franz Korf
Zweitgutachter : Prof. Dr. Gunter Klemke

Abgegeben am 23. September 2008

Arne Saggau

Thema der Bachelorarbeit

Sicherheitsorientierte Weiterentwicklung der mikrocontrollerbasierten Telemetrie in einem Formula Student Rennwagen durch Migration von TTCAN zu FlexRay

Stichworte

FlexRay, Verteiltes eingebettetes Echtzeitsystem, Mikrocontroller, zeitgesteuerte Kommunikation, statische Taskplanung, TTCAN, Telemetrie

Kurzzusammenfassung

Diese Bachelorarbeit beschreibt die Entwicklung eines verteilten eingebetteten Echtzeitsystems, das auf der Basis des Kommunikationssystems FlexRay dem sicherheitsorientierten Fortschritt der Automobilindustrie folgt. Die vorteilhaften Eigenschaften von FlexRay werden genutzt, um die Datenerfassung in einem Formula Student Rennwagen zuverlässig durchzuführen und die Grundlage zu schaffen, um das Kommunikationssystem für zukünftige sicherheitsrelevante Steuerungsaufgaben einsetzen zu können. Die Realisierung der verteilten Subsysteme wird unter Verwendung zweier verschiedener Echtzeitbetriebssysteme durchgeführt, die für die zeitgesteuerte Taskaktivierung und die Synchronisation der lokalen Uhren mit der vom Kommunikationssystem zur Verfügung gestellten globalen Uhrzeit eingesetzt werden.

Arne Saggau

Title of the paper

Safety-driven further development of the microcontroller based telemetry in a Formula Student racing car by migration of TTCAN to FlexRay

Keywords

FlexRay, distributed embedded real-time system, microcontroller, time-triggered communication, static scheduling dispatcher, TTCAN, telemetry

Abstract

This bachelor's thesis describes the development of a distributed embedded real-time system based on the communications system FlexRay resulting from the safety-driven improvement of the automotive industry.

The beneficial features of FlexRay are used to enable reliable data logging in a Formula Student race car, establishing a basis to use the communications system for future security-relevant control tasks.

The implementation of the distributed subsystems is carried out using two different real-time operating systems which are applied to enable the time triggered task activation and synchronization of the local clocks with the global time provided by the communications system.

Inhaltsverzeichnis

1. Einleitung

Die Anzahl der elektronischen Steuergeräte im Automobil hat in den letzten Jahren stark zugenommen. Neben den in der Automobilindustrie etablierten Technologien wie zum Beispiel das Antiblockiersystem, sollen in zukünftigen Fahrzeuggenerationen X-by-Wire-Systeme, wie Brake-by-wire und Steer-by-wire folgen. Für die Realisierung solcher sicherheitskritischer Systeme müssen die beteiligten Komponenten den Eigenschaften eines harten Echtzeitsystems gerecht werden. Das heißt, dass die benötigten Daten solcher Regelungs- und Steuerungsfunktionen nicht nur zuverlässig zwischen den Komponenten ausgetauscht werden, sondern auch innerhalb einer festgelegten Zeitspanne eintreffen müssen.
Daraus resultieren folgende Anforderungen an das für die Datenübertragung verwendete Kommunikationssystem [Wiew2004]:

- Hohe Übertragungsgeschwindigkeit
- Hohe Verfügbarkeit
- Hohe Zuverlässigkeit
- Deterministisches Verhalten

Da der CAN-Bus (Controller Area Network) als bisheriges Standard-Kommunikationssystem der Automobilindustrie auch aufgrund der relativ niedrigen Übertragungsgeschwindigkeit die Anforderungen nicht erfüllen kann, wurden in den letzten Jahren unterschiedliche Bussysteme entwickelt. Eine dieser Entwicklungen ist das Protokoll und Bussystem FlexRay. Neben den oben aufgeführten Eigenschaften wurden beim Entwurf der FlexRay-Architektur noch weitere technische und wirtschaftliche Ziele festgelegt, die aus den Erfahrungen mit den in Kraftfahrzeugen eingesetzten Kommunikationsprotokollen abgeleitet worden sind.
Eines dieser technischen Ziele, das bei der Entwicklung erreicht wurde, ist die Verfügbarkeit von zwei physikalischen Kommunikationskanälen, die je nach Anwendung entweder für erhöhte Verlässlichkeit sorgen, indem sie zur redundanten Datenübertragung verwendet werden oder bei nicht sicherheitskritischen Anwendungen für eine Verdopplung des Datendurchsatzes genutzt werden können [Schu2006].
In dieser Bachelorarbeit werden die sicherheitsorientierten Eigenschaften von FlexRay genutzt, um die Verlässlichkeit eines eingebetteten Systems in einem Formula Student Rennwagen zu erhöhen. Formula Student ist ein Projekt in dem Studenten in Teamarbeit einen einsitzigen Rennwagen mit dem Ziel entwerfen, bei einem Wettbewerb gegen Teams aus der ganzen Welt anzutreten. Bei diesem Wettbewerb gewinnt nicht das schnellste Auto, sondern das Team mit dem besten Gesamtpaket aus Konstruktion, Rennperformance, Finanzplanung und Verkaufsargumenten [Form2008].

Zum einen umfasst diese Arbeit die Migration einer, in vorangegangenen Bachelorarbeiten entwickelten Systemarchitektur, die auf den Bussystem CAN basierend für die telemetrische Datenerfassung im Rennwagen eingesetzt wird. Die Daten werden hierbei durch Sensoren erfasst, die an den entsprechenden, räumlich getrennten Stellen im Fahrzeug angebracht sind und anschließend ausgewertet werden, um die Fahreigenschaften und die Konstruktion verbessern zu können. Das Kommunikationssystem wird eingesetzt, um den Nachrichtenaustausch zwischen den verteilten Komponenten zu ermöglichen und die ermittelten Daten so an eine zentrale Komponente übertragen zu können, die diese über eine Schnittstelle direkt visualisieren und für eine spätere Auswertung aufzeichnen kann.
Zum anderen wird durch die Umstellung auf ein FlexRay-Kommunikationssystem die Grundlage für den zukünftigen Einsatz von Fahrerassistenzsystemen geschaffen. Im Gegensatz zur Telemetrie ist es für die Realisierung solcher sicherheitskritischer Systeme, die beispielsweise eine Antriebsschlupfregelung oder eine elektronische Stabilisierung des Fahrzeugs ermöglichen, zwingend erforderlich, dass mit dem eingesetzten Bussystem eine sehr zuverlässige und fehlertolerante Datenübertragung umgesetzt werden kann.

1.1 Aufbau der Bachelorarbeit

Die Arbeit gliedert sich in sechs Kapitel.

Im folgenden Kapitel werden verschiedene im Automobilbereich verwendete Kommunikationssysteme und die zugehörigen Netzwerkprotokolle erläutert. Dabei liegt das Hauptaugenmerk auf dem Kommunikationsprotokoll CAN und dem darauf aufsetzenden Time Triggered CAN (TTCAN), da die in vorangegangenen Bachelorarbeiten [Schu2007, Haas2007] entworfene und zu migrierende Systemarchitektur darauf aufgebaut ist. Des Weiteren werden die dabei eingesetzten Buszugriffsverfahren erläutert und die Vorteile, die sich durch die Migration zu FlexRay ergeben.

Im Kapitel 3 wird die Funktionsweise des Kommunikationssystems FlexRay beschrieben und es wird hervorgehoben, welche Möglichkeiten durch das Protokoll gegeben sind, um die Kommunikation in einem verteilten System, in bezug auf die Verlässlichkeit, zu verbessern.

Kapitel 4 beinhaltet das Design und die Realisierung des Gesamtsystems, wobei die allgemeinen Systemanforderungen definiert und die daraus resultierenden Aktivitäten bestimmten Subsystemen zugeordnet werden. Außerdem wird der Ablaufplan entworfen, der den Buszugriff zwischen den Subsystemen zeitlich koordiniert und als Grundlage für die Konfiguration des FlexRay-Protokolls dient.

Im Kapitel 5 folgen der Entwurf und die Implementierung der Subsysteme. Dafür werden die im vorigen Kapitel für jedes Subsystem spezifizierten Aufgaben durch die Implementierung von Tasks realisiert. Die Ausführungsreihenfolge und die Zeitpunkte der Taskaktivierung werden festgelegt und mit Hilfe von zwei verschiedenen Betriebssystemen sichergestellt. Abschließend werden die Lösungsansätze verglichen und bewertet.

Kapitel 6 stellt eine Zusammenfassung dieser Arbeit dar und zeigt auf, wo mögliche Weiterentwicklungen in der Zukunft ansetzen könnten.

Im Anhang der vorliegenden Arbeit werden die Installation und die Verwendung der eingesetzten Soft- und Hardware beschrieben, womit die Grundlage für einen schematischen Einstieg bei fortführenden Projekten gegeben ist.

2. Bussysteme im Automobil

In diesem Kapitel werden verschiedene im Automobilbereich verwendete Kommunikationssysteme und die dabei eingesetzten Netzwerkprotokolle erläutert (vgl. Abbildung 1). Insbesondere wird das weitverbreitete Bussystem CAN (Controller Area Network) beschrieben, um im darauffolgenden Kapitel aufzeigen zu können, welche Vorteile und Möglichkeiten durch die Migration zu einem FlexRay-System gegeben sind.

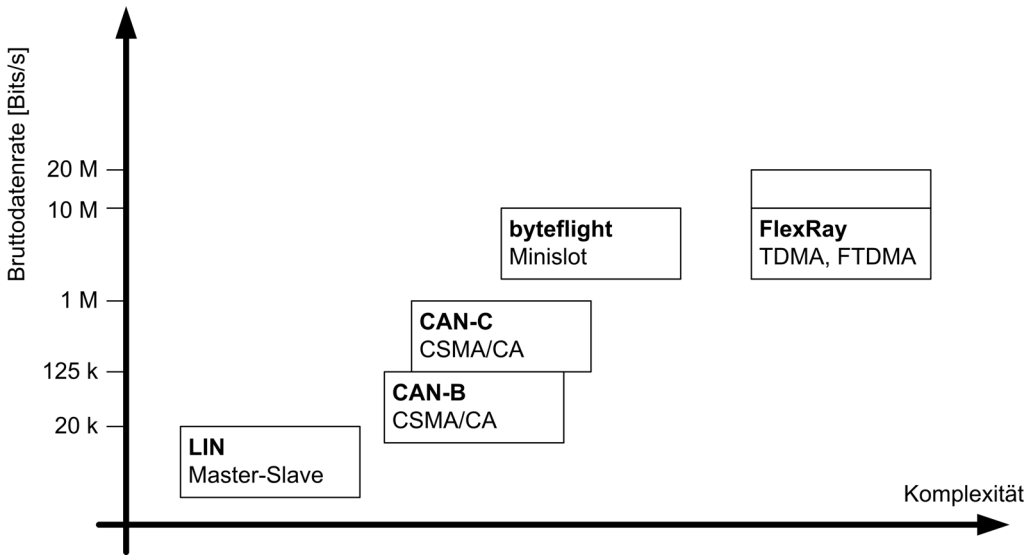

Abbildung 1 – Kommunikationsprotokolle im Automobil [Raus2008]

2.1 Buszugriffsverfahren

Kommunikationssysteme werden verwendet, um den Austausch von Nachrichten zwischen verteilten Hardware-Komponenten zu ermöglichen. So können Abläufe koordiniert und zum Beispiel komplexe Steuerungsaufgaben, die von den Eingangsgrößen mehrerer Komponenten abhängig sind, durchgeführt werden.
Die Nachrichten werden dabei über einen oder mehrere Kommunikationskanäle ausgetauscht, wobei ein Kanal eine gemeinsame Ressource darstellt, die zu einem Zeitpunkt immer nur von einem Kommunikationsteilnehmer genutzt werden kann, sodass es erforderlich ist, Regeln für die Nutzung festzulegen. Bei Kommunikationssystemen wird in diesem Zusammenhang von Zugriffssteuerung oder Zugriffsverfahren gesprochen. Dabei kann, bezüglich der Art und Weise wie auf den Bus zugegriffen wird, zwischen deterministischen und zufälligen Verfahren unterschieden werden. Die Leistungsfähigkeit eines Bussystems in bezug auf Latenzzeit, Echtzeitverhalten und Fehlertoleranz hängt entscheidend von dem eingesetzten Zugriffsverfahren ab [Raus2008][Geva2005][Etsc2002].

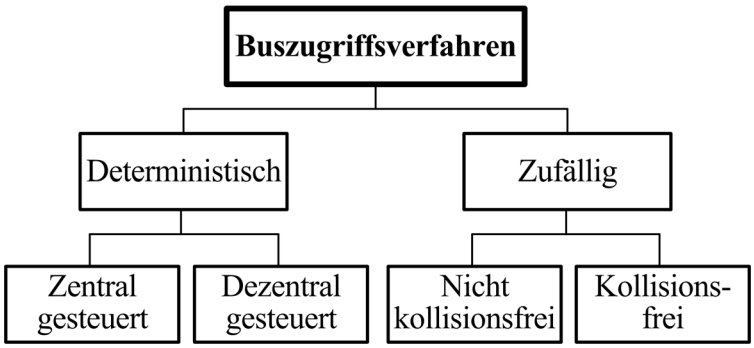

Abbildung 2 – Einteilung von Buszugriffsverfahren

2.1.1 Deterministische Zugriffsverfahren

Bei den deterministischen Verfahren handelt es sich um einen kontrollierten Vorgang, der den gleichzeitigen Zugriff mehrerer Teilnehmer ausschließt und es ermöglicht, die Antwortzeiten vorherzusagen. Hierbei wird die Vergabe des Buszugriffsrechts, wie in Abbildung 2 dargestellt, entweder zentral oder dezentral gesteuert.

Zentral gesteuerte Zugriffsverfahren
Bei zentral gesteuerten Verfahren wird der exklusive Buszugriff durch einen übergeordneten Teilnehmer geschützt, der anhand eines festgelegten Zeitplans und einer festgelegten Reihenfolge den anderen Teilnehmern die Freigabe zum Senden erteilt, sodass definierte Latenzzeiten gegeben sind. Zentral gesteuerte Zugriffsverfahren sind einfach zu realisieren, aber der Ausfall des übergeordneten Teilnehmers führt zum Ausfall der gesamten Kommunikation, sodass je nach Anforderung an die Verfügbarkeit des Systems die redundante Auslegung dieser Komponente notwendig sein kann [Etsc2002].

Dezentral gesteuerte Zugriffsverfahren
Bei den dezentral gesteuerten Zugriffsverfahren werden jedem Teilnehmer feste Zeitabschnitte zugeordnet, zu denen er das alleinige Recht besitzt auf den Bus zuzugreifen. Dieses Verfahren wird daher als Time Division Multiple Access (TDMA) bezeichnet und ermöglicht die Erstellung eines Ablaufplan, in dem die zeitliche Aufteilung der verfügbaren Bandbreite festgehalten wird. Die Voraussetzung für die Einhaltung des Ablaufplans ist ein hohes Maß an Synchronität zwischen den lokalen Uhren aller Kommunikationsteilnehmer. Dafür können Synchronisationsalgorithmen durch Hardware implementiert oder direkt im Protokoll, das für die Nachrichtenübertragung verwendet wird, spezifiziert werden.
Der Vorteil des TDMA-Verfahrens ist die Vorhersagbarkeit des Buszugriffs jedes Kommunikationsteilnehmers und dass somit garantierte maximale Latenzzeiten für die Nachrichtenübertragungen gegeben sind [Knol2006] [Etsc2002].

2.1.2 Zufällige Zugriffsverfahren

Bei den zufälligen Zugriffsverfahren ist die Möglichkeit gegeben, dass mehrere Kommunikationsteilnehmer den Bus nutzen wollen und zeitgleich mit dem Senden beginnen, sobald dieser als frei erkannt wird. Diese Verfahren werden daher als Carrier Sense Multiple Access (CSMA) bezeichnet. Je nach Auflösung der, durch den zeitgleichen Zugriff hervorgerufenen Konflikte, unterscheidet man zwischen kollisionsfreien und nicht kollisionsfreien Verfahren. Bei den zufälligen Zugriffsverfahren kann im Allgemeinen keine Aussage über die Sendezeitpunkte bestimmter Teilnehmer getroffen werden, sodass kein deterministisches Verhalten vorliegt.

Nicht kollisionsfreie Zugriffsverfahren
Die nicht kollisionsfreien Verfahren sind darauf ausgelegt, die durch gleichzeitige Sendevorgänge auftretenden Konflikte zu erkennen und werden daher als CSMA/CD (Collision Detect) bezeichnet. Die Teilnehmer dieses Verfahrens detektieren die Kollisionen durch die Überprüfung des Buspegels während der Übertragung oder durch die Auswertung der Datensicherungssequenz nach Abschluss eines Sendevorgangs.
Die Erkennung einer Kollision führt zur Beendigung aller Übertragungen und zu dem Versuch der Teilnehmer, die Nachricht zu einem späteren, teilnehmerspezifischen Zeitpunkt erneut zu senden [Reif2007] [Etsc2002].

Kollisionsfreie Zugriffsverfahren
Bei den kollisionsfreien Verfahren, die sogenannten CSMA/CA-Verfahren (Collision Avoidance), ist es ebenfalls möglich, dass mehrere Kommunikationsteilnehmer gleichzeitig mit dem Senden ihrer Nachrichten beginnen, wobei Kollisionen hier durch die Vergabe von Prioritäten grundsätzlich vermieden werden. Die gesendeten Nachrichten werden dazu mit einer Kennung versehen, die auch als Identifier (ID) bezeichnet wird und die Priorität der Nachricht bestimmt. Der gleichzeitige Buszugriff kann so durch die Teilnehmer bereits vor der Übertragung der eigentlichen Nachricht innerhalb einer Arbitrierungsphase erkannt werden. Während dieser Phase wird die Nachrichten-ID von den gleichzeitig sendenden Teilnehmern bitweise auf den Bus gelegt. Durch das Zurücklesen des Bus-Pegels kann jeder Teilnehmer feststellen, ob die von ihm gesendete Kennung durch eine höherpriore Kennung überlagert wurde. In diesem Fall zieht sich der Teilnehmer zurück, sodass nach dem Abschluss der Arbitrierungsphase nur der Sender mit der zu diesem Zeitpunkt höchstprioren Nachricht auf dem Bus verbleibt. Die unterlegenden Teilnehmer starten einen erneuten Versuch, sobald der Bus wieder frei ist. [Etsc2002] [Geva2005].

2.2 Controller Area Network

CAN wurde 1981 von der Firma Bosch für die Vernetzung der verteilten Systeme in Automobilen entwickelt und kommt heute in Verbindung mit zusätzlichen standardisierten höheren Protokollspezifikationen auch in zahlreichen anderen Anwendungsbereichen zum Einsatz.
Das CAN zugrundeliegende Standard-Protokoll ist für ereignisgesteuerte, nicht-deterministische Software konzipiert worden und ist daher mit dem zufälligen Buszugriffsverfahren CSMA/CA für den Nachrichtenaustausch realisiert worden. Die für dieses Verfahren eingesetzten Nachrichtenkennungen können außerdem zur Filterung verwendet werden, sodass die Kommunikationsteilnehmer nur die für sie relevanten Nachrichten empfangen.
Für die Länge der Nachrichtenkennung sind durch die CAN-Spezifikation zwei miteinander kompatible Nachrichtenformate definiert, das Standard-Format mit 11-Bit Identifier und das Extended Format mit 29-Bit Identifier. Die maximale Länge der Nutzdaten einer CAN-Nachricht ist bei beiden Formaten auf 8 Byte begrenzt, sodass die Übertragung größerer Datenblöcke die Aufteilung auf mehrere Nachrichten erfordert.
Für die Feststellung von Übertragungsfehlern enthält jede CAN-Nachricht ein Prüfsummenfeld, das für die zyklische Redundanzprüfung verwendet wird.
Ein weiterer durch das CAN-Protokoll vorgesehener Sicherheitsmechanismus ist die Erkennung defekter Kommunikationsteilnehmer. Ein Teilnehmer, der den Datenverkehr auf dem Bus stört, wird beim Überschreiten einer festgelegten Fehlerrate zunächst mit beschränkten Sendemöglichkeiten belegt und bei weiteren Störungen vom Netzwerk getrennt.
Aufgrund der vielfältigen Anforderungen werden CAN-Busse mit unterschiedlichen Datenraten eingesetzt, um die dem jeweiligen Anwendungsbereich entsprechenden Netzausdehnungen zu ermöglichen. Die bei einer bestimmten Datenrate maximal mögliche Entfernung zwischen den Endpunkten eines Netzwerks ist maßgeblich durch die auf der physikalischen Verbindung erforderliche Signallaufzeit begrenzt. In ISO-Normen definierte Ausführungen sind CAN-B und CAN-C, wobei CAN-B auf größere Netzausdehnungen bei geringer Datenrate ausgelegt ist und CAN-C eine hohe Geschwindigkeit für kleinere Entfernungen zur Verfügung stellt [Etsc2002] [Pree2007].

2.3 Time-Triggered CAN

Durch das im CAN-Protokoll verwendete CSMA/CA-Verfahren ist die Möglichkeit zur deterministischen Datenübertragung stark begrenzt. Für sicherheitsrelevante Anwendungen mit hohen Echtzeitanforderungen ist das Buszugriffsverfahren nach dem TDMA-Prinzip erforderlich, da so die Latenzzeiten aller Nachrichten kalkulierbar sind und damit ein zeitdeterministisches Verhalten gegeben ist.
Für einen solchen Ansatz stellt Time-Triggered CAN (TTCAN) ein übergeordnetes zeitgesteuertes Protokoll zur Verfügung, welches auf der CAN-Protokoll Spezifikation basiert.
Die Steuerung des zeitlichen Ablaufs in einem TTCAN-Netzwerk übernimmt ein Netzwerkknoten, der als Time Master bezeichnet wird. Durch diesen Knoten werden periodisch Referenznachrichten gesendet, anhand derer die übrigen Kommunikationsteilnehmer ihre lokalen Uhren synchronisieren können [Reif2007].
Abbildung 3 zeigt einen möglichen zeitlichen Ablaufplan der Kommunikation eines TTCAN-Netzwerks und verdeutlicht die periodische Übertragung der Referenznachrichten zu Beginn der sogenannten Basiszyklen.

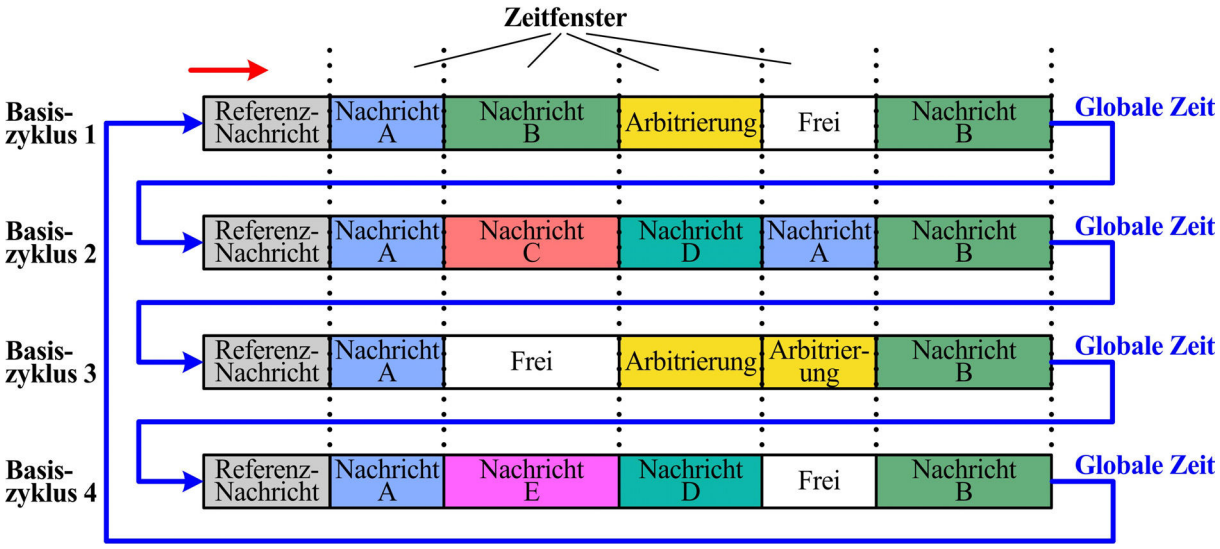

Abbildung 3 - Aufbau eines TTCAN Matrixzyklus

Ein Basiszyklus besteht aus mehreren Zeitfenstern, wobei zwischen exklusiven, arbitrierenden und freien Zeitfenstern unterschieden wird.
Die exklusiven Zeitfenster stellen die beschriebenen Abschnitte dar, zu denen jeweils ein Netzwerkknoten das alleinige Buszugriffsrecht erhält und die deterministische Datenübertragung ermöglicht wird. Beim Design eines TTCAN-Netzwerks können innerhalb eines Basiszyklus mehrere exklusive Zeitfenster für einen Netzwerkknoten und die von ihm gesendeten Nachrichten festgelegt werden.

Die arbitrierenden Zeitfenster dienen der ereignisgesteuerten Übertragung nach dem CSMA/CA-Verfahren, sodass durch die zerstörungsfreie, bitweise Arbitrierung die höchstpriore Nachricht übertragen wird.
Freie Zeitfenster können in den Basiszyklen für zukünftige Erweiterungen eingeplant werden, wodurch die Integration neuer Kommunikationsteilnehmer erleichtert wird oder bereits vorhandenen Knoten mehr Bandbreite zur Verfügung gestellt werden kann.
Da die Anzahl und die Länge der verschiedenen Zeitfenster bereits beim Entwurf eines Netzwerks festgelegt werden und so durch einen einzelnen Basiszyklus nicht genügend Flexibilität gegeben ist, unterstützt TTCAN die Verbindung mehrerer Basiszyklen zu einem sogenannten Matrixzyklus. Dadurch besteht die Möglichkeit den Zeitfenstern der Basiszyklen unterschiedliche Nachrichten zuzuweisen und auch bei der Art der Zeitfenster zwischen exklusiv, frei und arbitrierend wechseln zu können [Bosc2000].

Durch das Aufsetzen auf dem Standard CAN-Protokoll sind die in Kapitel 2.2 beschriebenen Sicherheitsmechanismen auch in einem auf TTCAN basierenden Kommunikationssystem gegeben. Die durch das TTCAN-Protokoll vorgesehene, übergeordnete Funktion des Time Masters stellt hingegen eine Gefährdung für die Verlässlichkeit eines solchen Systems dar. Bei einem Ausfall des Time Masters wäre die gemeinsame Zeitbasis der Teilnehmer, die durch den Versand der Referenznachrichten etabliert wird, nicht mehr gegeben und die deterministische Kommunikation nicht mehr gewährleistet. Daher ist es gegebenenfalls erforderlich, diese zentrale Komponente redundant auszulegen, um dem Fehlerfall entgegenzuwirken.

2.4 Local Interconnect Network

Local Interconnect Network (LIN) ist ein kostengünstiges Kommunikationssystem mit geringen Datenübertragungsraten, das für Einsatzbereiche konzipiert wurde, in denen die Leistungsfähigkeit eines CAN-Netzwerks nicht erforderlich ist und keine sicherheitskritischen Anforderungen zu erfüllen sind.
Die Kommunikation basiert auf dem Master-Slave-Prinzip, bei dem der gleichzeitige Buszugriff mehrerer Teilnehmer durch die Koordination eines übergeordneten Knotens verhindert wird. Dieser als Master bezeichnete Netzwerkteilnehmer übernimmt die Initiierung der Kommunikation, indem er die anderen Knoten, die sogenannten Slaves, zum Senden ihrer Daten auffordert.
Die Umsetzung erfolgt über zweiteilige Nachrichten, die jeweils aus einem Header- und einem Response-Teil bestehen. Der Header enthält einen eindeutigen Identifier und wird vom Master an alle Slaves versendet, worauf der Knoten, dessen Identifier mit dem des Headers übereinstimmt, mit dem Response-Teil der Nachricht antwortet [LIN2006] [Pree2007].
In Abbildung 4 ist der Ablauf dieses Verfahren exemplarisch dargestellt.

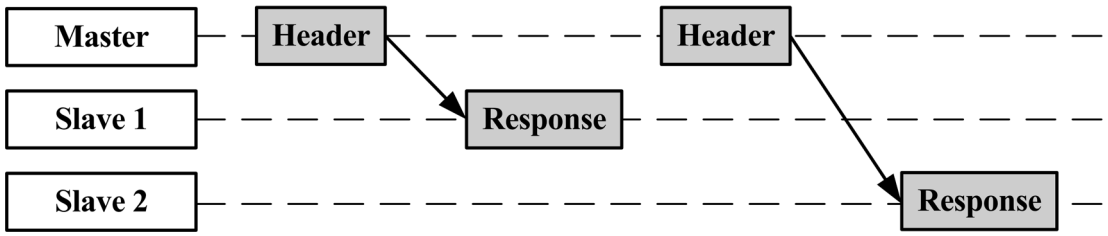

Abbildung 4 – Buszugriffssteuerung im Local Interconnect Network

Die Reihenfolge, in der die Slaves durch den Master zum Senden aufgefordert werden, ist durch eine vorab festgelegte Tabelle spezifiziert, die periodisch abgearbeitet wird. In LIN kann der Master mehrere Tabellen, die auch als Schedule bezeichnet werden, speichern und unter diesen auswählen, sodass die Kommunikationsreihenfolge dynamisch geändert werden kann.
Neben diesem zeitgesteuerten Ansatz wird durch das LIN-Protokoll die ereignisgesteuerte Kommunikation unterstützt, wobei die Kollisionsvermeidung durch das CSMA-Verfahren erreicht wird. Die geringe Bandbreite kann so effektiver genutzt werden, da die Slaves nicht regelmäßig zum Senden aufgefordert werden, sondern nur auf den Bus zugreifen, wenn Änderungen aufgetreten sind und neue Daten vorliegen.
Vorkehrungen, die der Fehlertoleranz dienen, sind in einem LIN-Kommunikationssystem auf eine CRC-Prüfsumme der maximal acht Byte Nutzdaten und auf die Sicherung der Identifier durch ein Parity-Bit begrenzt [Reif2007] [Pree2007].

2.5 byteflight

Byteflight ist für Anwendungen mit sicherheitsrelevanten Anforderungen entwickelt worden und ermöglicht eine flexible Konfiguration für die optimale Nutzung der verfügbaren Bandbreite.
Im byteflight-Protokoll ist dafür der zeitgesteuerte Buszugriff umgesetzt worden, der durch die zeitliche Aufteilung der Bandbreite koordiniert wird.
Dem TDMA-Verfahren entsprechend, werden den Kommunikationsteilnehmern dafür Zeitschlitze zugeteilt, zu denen der exklusive Buszugriff garantiert wird. Da bei dem realisierten Verfahren des byteflight-Protokolls diese Zeitschlitze von variabler Länge sein können und von den Teilnehmern nur im Bedarfsfall für die Datenübertragung der maximal zwölf Byte Nutzdaten verwendet werden, wird es als Flexible Time Division Multiple Access (FTDMA) bezeichnet.
Die konkrete Umsetzung in einem byteflight-Kommunikationssystem erfolgt auf Basis von zyklischen Synchronisierungspulsen, durch die das erforderliche gemeinsame Zeitverständnis der Netzwerkknoten sichergestellt wird. Die Synchronisierungspulse werden im festgelegten Abstand von 250 Mikrosekunden durch einen ausgewählten Kommunikationsteilnehmer, der als sogenannter Sync-Master konfiguriert ist, an alle Teilnehmer gesendet.
Diese feststehende Periodendauer mit der die Synchronisierungspulse wiederholt werden entspricht der Zykluszeit, die in Zeitschlitze unterteilt für die Nachrichtübertragung zur Verfügung steht. Des Weiteren ist jedem Zeitschlitz eine Identifikationsnummer zugewiesen und mit dem Beginn jeder Periode inkrementiert jeder Netzwerkknoten einen Zähler mit dem Wert Null beginnend bis zur höchstmöglichen Identifikationsnummer.
Sobald der Zähler einen Wert annimmt, der gleich dem Identifier einer Nachricht ist, die gesendet werden soll, wird die Datenübertragung gestartet und alle Knoten warten mit der Erhöhung ihrer Zähler, bis diese abgeschlossen ist [Grie2000] [Pree2007].

In Abbildung 5 wird dieser Vorgang exemplarisch dargestellt und außerdem die möglichen Konfigurationen, die durch das byteflight-Protokoll unterstützt werden, verdeutlicht.

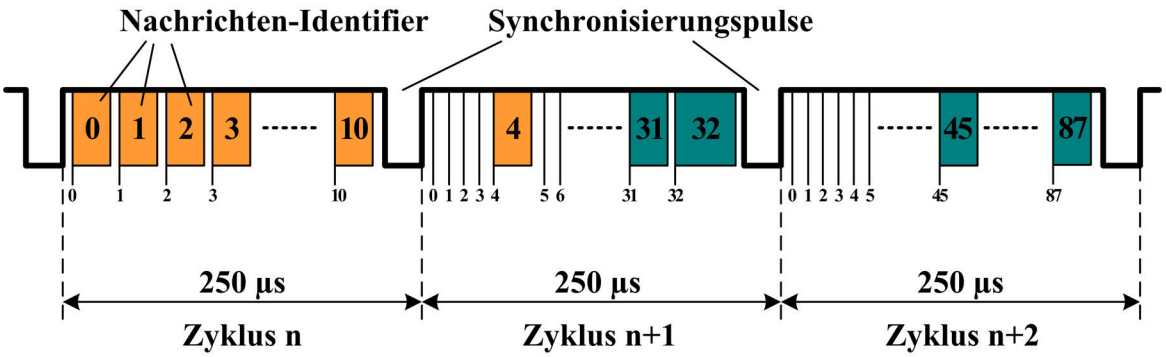

Abbildung 5 – FTDMA-Verfahren im byteflight-System

Im ersten Zyklus wurde die zugrundeliegende Konfiguration beim Entwurf des Systems so ausgelegt, dass die gesamte Bandbreite für die Übertragung der höchstprioren Nachrichten benötigt wird. Für eine Anwendung, deren Anforderungen die Übertragung einer festen Anzahl von Nachrichten mit garantierten Latenzzeiten erfordert, kann ein byteflight-System mit einer solchen Konfiguration verwendet werden. Die maximal mögliche Wiederholungsrate ist dabei durch die festgelegte Zykluszeit begrenzt, sodass die Übertragung der Nachrichten frühestens nach 250 Mikrosekunden wiederholt werden kann.
Hingegen wird im zweiten Zyklus des dargestellten Beispiels lediglich eine hochpriore Nachricht übertragen und die übrige Bandbreite zum Absetzen niederpriorer Nachrichten eingesetzt.
Im dritten dargestellten Zyklus können noch Nachrichten mit relativ hohen Identifikationsnummern abgesetzt werden, da in diesem Fall keine Sendeanforderungen für höherpriore Nachrichten vorliegen.
Die Vergabe der Prioritäten über die Nachrichten-Identifier und die Frequenz der Sendeanforderung bestimmter Nachrichten sind somit die Faktoren, die bei einem Entwurf entscheidenden Einfluss auf das spätere Verhalten des byteflight-Kommunikationssystems haben.

Für die Fehlererkennung und Fehlerbehandlung ist im byteflight-Protokoll die zyklische Redundanzprüfung vorgesehen und die Möglichkeit einen ausgefallenen Sync-Master durch einen anderen dafür konfigurierten Teilnehmer zu ersetzen.

3. FlexRay

In diesem Kapitel wird das Protokoll und Bussystem FlexRay beschrieben.
FlexRay ist die Entwicklung eines Konsortiums, in dem sich viele namhafte Firmen der Automobil- und Halbleiterindustrie zusammengeschlossen haben.
Die bei der Entwicklung verfolgten Ziele waren auf ein fehlertolerantes und deterministisches Bussystem ausgelegt, dass auch für harte Echtzeitanforderungen und somit für sicherheitskritische Anwendungen geeignet ist [Raus2008] [Geva2005].
In den folgenden Kapiteln werden die Eigenschaften von FlexRay sowie das Protokoll näher erläutert und insbesondere auf die Vorteile von FlexRay in bezug auf die Verlässlichkeit und die damit verbundene Fehlertoleranz eingegangen. Für eine vollständige Beschreibung sämtlicher Details des Protokolls wird auf [Flex2004] und [Raus2008] verwiesen.

3.1 Eigenschaften von FlexRay

Das FlexRay-Kommunikationssystem unterstützt zwei voneinander unabhängige Kommunikationskanäle mit einer Datenrate von je 10Mbit pro Sekunde.
Zum einen können so zwei unterschiedliche Nachrichten zur gleichen Zeit übertragen und die Datenrate verdoppelt werden, womit FlexRay im Vergleich zu anderen Kommunikationsprotokollen eine hohe Übertragungsgeschwindigkeit zur Verfügung stellt.
Zum anderen kann der zweite Kanal auch zur redundanten Nachrichtenübertragung verwendet werden, um speziell in sicherheitskritischen Anwendungsbereichen Fehlertoleranz bei der Kommunikation zu erreichen [Schu2006].
Die im Protokoll umgesetzten Buszugriffsverfahren TDMA (vgl. Kapitel 2.1.1) beziehungsweise FTDMA (vgl. Kapitel 2.5) ermöglichen den Kommunikationsteilnehmern zu einem bestimmten Zeitpunkt den exklusiven Zugriff auf die Kommunikationskanäle. Die so erreichte Echtzeitfähigkeit und damit verbundene Deterministik erlauben es, die Reaktionszeit des Kommunikationssystems zu kalkulieren.
Voraussetzung für den zeitgesteuerten Zugriff ist eine synchronisierte Zeitbasis aller an der Kommunikation beteiligten Netzwerkknoten. Diese Zeitbasis wird vom Protokoll selbstständig etabliert und synchron gehalten [Raus2008].

3.2 Aufbau eines FlexRay-Knotens

Ein FlexRay-Knoten, wie in Abbildung 6 gezeigt, stellt einen von mehreren Kommunikationsteilnehmern eines FlexRay-Netzwerks dar.

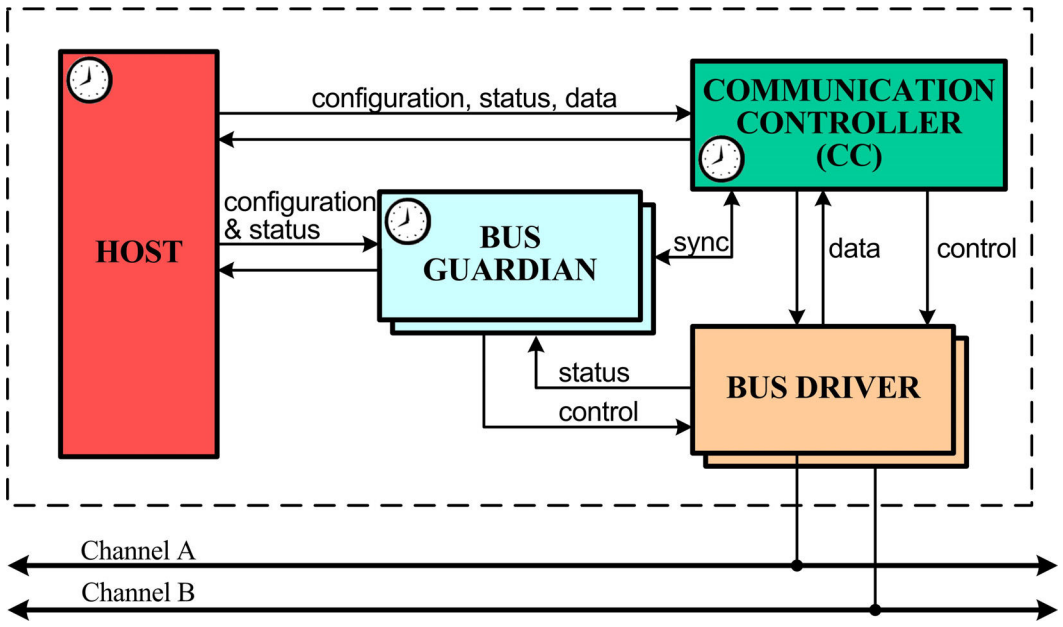

Abbildung 6 – Aufbau eines FlexRay-Knotens [Szec2007]

Ein Knoten besteht aus einem Mikrocontroller, kurz Host genannt, auf dem das eigentliche Anwendungsprogramm ausgeführt wird. Das FlexRay-Protokoll wird durch den Kommunikationscontroller realisiert, der dem Host eine Schnittstelle zur Verfügung stellt, über die der Austausch der zu sendenden und zu empfangenden Daten stattfindet und über die der Host die Konfiguration des Kommunikationscontrollers durchführen kann [Reif2007].
Eine weitere Schnittstelle verbindet den Kommunikationscontroller mit den Bustreibern (engl. Bus Driver), die die physikalische Verbindung zu den FlexRay-Kanälen herstellen.
Zusätzlich können Buswächter(engl. Bus Guardian) eingesetzt werden, die die Sicherheit des Netzwerks erhöhen, indem sie die Kommunikation kontrollieren und den Schreibzugriff defekter Teilnehmern auf die FlexRay-Kanäle sperren.

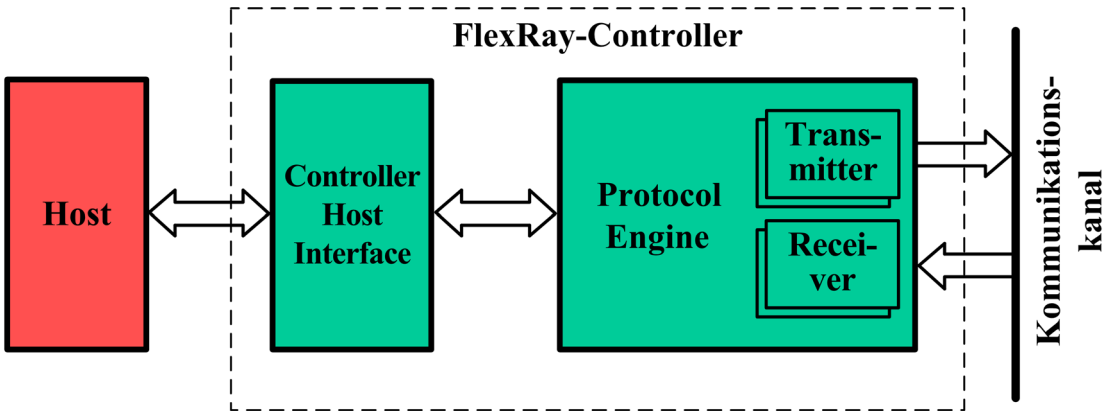

Abbildung 7 – Aufbau eines FlexRay-Kommunikationscontrollers [Raus2008]

Der interne Aufbau des Kommunikationscontrollers ist in Abbildung 7 dargestellt. Die bereits beschriebene Schnittstelle zum Mikrocontroller wird als Controller-Host-Interface bezeichnet. Durch die Protocol Engine werden die Kern-Mechanismen des FlexRay-Protokolls realisiert [Geva2005]. Einen Teil der Protocol Engine bilden Transmitter und Receiver, die für das Codieren bzw. Decodieren der Signale zuständig sind, die vom Bustreiber gesendet bzw. empfangen werden.
Des Weiteren werden die folgenden Mechanismen von der Protocol Engine umgesetzt:

- Synchronisation der lokalen Uhr
- Fehler-Management
- Starten des Netzwerks
- Generierung der Zeiteinheit gemäß der Zeitbasis des Netzwerks
- Prüfung der Gültigkeit von eingehenden Nachrichten

3.3 Bus Guardian

Der Bus Guardian ist ein autonomes Bauteil, das Bestandteil eines FlexRay-Knotens sein kann und das die Aufgabe hat, den Zugriff auf die Kommunikationskanäle zu kontrollieren und zu sichern.
Der Kommunikationscontroller eines FlexRay-Knotens kann theoretisch über den Bustreiber jederzeit auf den Kommunikationskanälen senden. Die Zeitpunkte der Übertragung werden jedem Knoten vorab durch den TDMA-Ablaufplan bekannt gemacht, sodass der exklusive Zugriff auf das Kommunikationsmedium gewährleistet ist und es zu keinen Kollisionen kommen kann, solange die Kommunikationscontroller fehlerfrei funktionieren [Raus2008], [Temp1998].
Der Bus Guardian wird für die Erhöhung der Sicherheit und der Fehlertoleranz eingesetzt, indem auch ihm der Ablaufplan des Buszugriffs bekannt gemacht wird und er anhand einer eigenen, mit dem Netzwerk synchronisierten Uhr, die Sendeaktivitäten des Kommunikationscontrollers überwacht. So hat der Bus Guardian das gleiche globale Zeitverständnis und kennt die genauen Sendezeitpunkte des FlexRay-Knotens, zu denen er dem Bustreiber über entsprechende Steuersignale die explizite Freigabe für die Ausführung des Sendevorgangs erteilt. Nach dem Abschluss der Übertragung wird der Sendepfad wieder gesperrt.
Bei einem Kommunikations- oder Synchronisationsfehler eines Kommunikationscontrollers wird so der gleichzeitige Zugriff mehrerer Controller auf die Kommunikationskanäle blockiert und so verhindert, dass ein fehlerhafter Knoten die gesendeten Frames anderer Knoten zerstören kann.
Zusätzlich kann der Bus Guardian zur Fehlererkennung eingesetzt werden. Dabei wird der Versuch eines Kommunikationscontrollers eine Nachricht zu senden, obwohl der Sendepfad gesperrt ist, durch den Buswächter festgestellt und dem Host signalisiert. Die danach durchgängige gesperrte Kommunikation des Controllers kann nur durch die entsprechende Fehlerbehandlung des Hosts wieder aufgenommen werden, die das Zurücksetzen und neu konfigurieren des Kommunikationscontrollers und des Bus Guardians vorsehen muss.
Da für den Zeitraum von der Sperrung bis zur erneuten Integration in das Netzwerk die geplante Funktion des betroffenen Teilnehmers nicht gegeben ist, ist in einem sicherheitskritischen System gegebenenfalls eine redundante Auslegung der Komponente erforderlich.

3.4 FlexRay-Kommunikationszyklus

Im FlexRay-Protokoll ist die Kommunikation in Zyklen unterteilt. Nach dem Ablauf eines Zyklus schließt sich, wie in Abbildung 8 dargestellt, direkt der nächste an. Jeder Zyklus hat eine Zyklusnummer, deren Zählung bei null beginnt und nach dem 63. Zyklus wieder auf null zurückgesetzt wird [Raus2008]. Der FlexRay-Kommunikationszyklus besteht aus den folgenden vier Bereichen, die in ihrer zeitlichen Größe, den Anforderungen entsprechend konfiguriert werden können:

- Statisches Segment
- Dynamisches Segment
- Symbol Window
- Network Idle Time

Das dynamische Segment und das Symbol Window sind optional und können bei dem Entwurf eines Systems, dessen Anforderungen eine rein zeitgesteuerte Kommunikation erfordern, unberücksichtigt bleiben [Reif2007]. Das bedeutet, dass die Konfiguration eines FlexRay-Systems so ausgelegt werden kann, dass sich der Zyklus nur aus dem statischen Segment und der Network Idle Time zusammensetzt. Diese beiden Bereiche müssen vorhanden sein, da sie für Methoden eingesetzt werden, die durch das FlexRay-Protokoll für die Uhrensynchronisation vorgesehen sind (vgl. Kapitel 3.6).

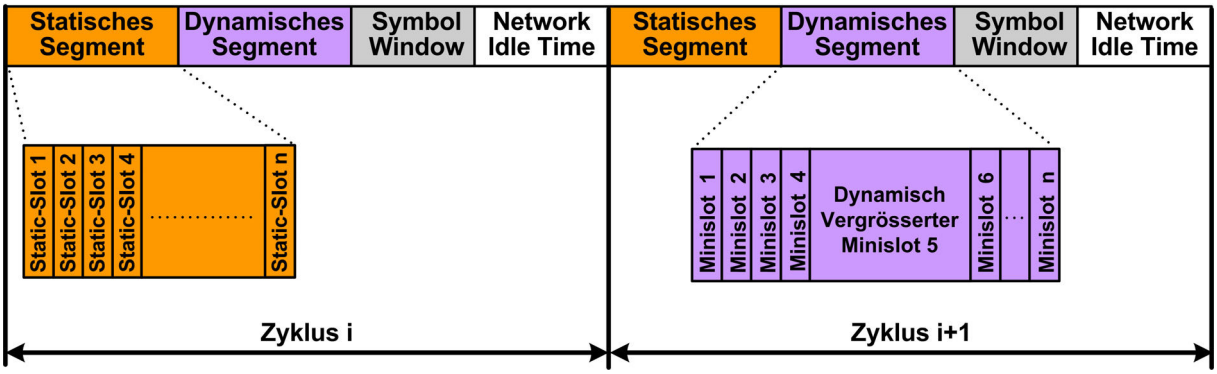

Abbildung 8 –FlexRay-Kommunikationszyklus [Temp2003]

Im statischen Segment erfolgt der Buszugriff nach dem TDMA-Verfahren. Das bedeutet, dass jedem Kommunikationsteilnehmer vorab ein oder mehrere Zeitfenster fest zugeordnet werden. Diese Zeitfenster werden als Static-Slots bezeichnet und garantieren jeweils den exklusiven Zugriff auf den Bus. Die aufsteigend nummerierten Static-Slots haben alle die gleiche zeitliche Länge und die darin gesendeten Nachrichten das gleiche Frame-Format (vgl. Kapitel 3.5).

Damit der Kommunikationszyklus bei allen Netzwerkknoten zeitgleich ausgeführt wird und der exklusive Zugriff gewährleistet bleibt, müssen die lokalen Uhren der Knoten miteinander synchronisiert werden [Raus2008].
Aus diesem Grund muss das statische Segment mindestens aus zwei Static-Slots bestehen. In diesen Slots werden spezielle Nachrichten gesendet, die für die Synchronisation der Kommunikationsteilnehmer verwendet werden. So kann eine globale Zeitbasis gebildet werden, die die Grundvoraussetzung für die Unterstützung von Echtzeitkommunikation im verteilten System ist, da so für jede Nachricht der Zeitpunkt der Übertragung vorausgesagt werden kann.

Das dynamische Segment ist in Minislots unterteilt, die wie die Zeitfenster des statischen Segments die gleiche Größe haben, aber im Gegensatz zu den Static-Slots eine kleinere zeitliche Länge aufweisen. Im dynamischen Segment können Nachrichten variabler Länge versendet werden, wobei der Minislot, in dem die Übertragung beginnt, solange um weitere Minislots vergrößert wird, bis der Sendevorgang der Nachricht abgeschlossen ist [Raus2008].
Ein weiterer Unterschied ist, dass die Kommunikationsteilnehmer nur im Bedarfsfall einen Sendevorgang starten. Falls ein Knoten in seinem zugeordneten Minislot keine Nachricht absetzen will, bleibt aufgrund der geringen Größe der Minislots auch der Anteil ungenutzter Bandbreite gering, da direkt im darauffolgenden Minislot ein anderer Kommunikationsteilnehmer eine Übertragung beginnen kann.
Zur Veranschaulichung dieses FTDMA-Verfahrens (vgl. Kapitel 2.5) sind in Abbildung 9 zwei Nachrichten dargestellt, wobei die eine Nachricht drei Minislots und die andere fünf Minislots für die Übertragung benötigt. In den übrigen Slots wurde von den ihnen zugeordneten Knoten nicht gesendet, sodass diese Slots genau der Größe eines Minislots entsprechen.

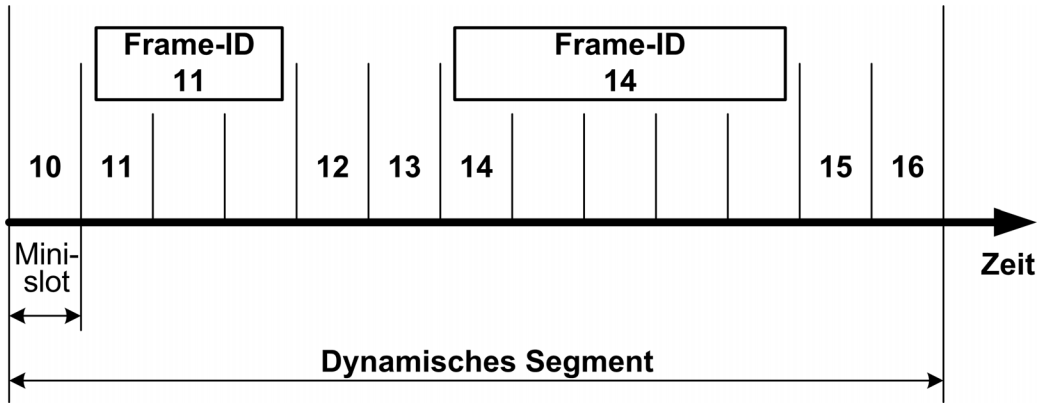

Abbildung 9 – Dynamisches Segment eines Kommunikationszyklus

Auch die Nummerierung, die im statischen Segment beginnt und im dynamischen Segment fortgeführt wird, ist davon abhängig wie viele Nachrichten gesendet werden, da der Zähler

erst inkrementiert wird, wenn die Übertragung einer Nachricht abgeschlossen ist.
Somit kann einem Knoten, dem der Minislot mit der kleinsten Nummer zugewiesen wird, die höchste Priorität für das Senden seiner Nachricht eingeräumt werden.
Dabei kann der Fall eintreten, dass ein Knoten eine Nachricht versendet, die sämtliche Minislots für die Übertragung in Anspruch nimmt und die Knoten im hinteren Bereich der Zuweisungsreihenfolge mit geringerer Priorität in diesem Zyklus ohne Sendemöglichkeit bleiben [Will2006]. Außerdem muss bei der FlexRay-Konfiguration ein Zeitpunkt in Form eines Minislots festgelegt werden, der in Abhängigkeit der Nachrichtenlänge den spätmöglichsten Beginn einer Nachrichtenübertragung im dynamischen Segment beschreibt. So kann sichergestellt werden, dass ein Knoten, der in diesem Minislot einen Sendevorgang beginnt, die Nachricht vollständig übertragen kann, bevor das Ende des dynamischen Segments erreicht ist.

Das dynamische Segment stellt somit das ereignisgesteuerte Kommunikationsmodell zur Verfügung, das eine flexible Nachrichtübertragung ermöglicht, aber keine Aussage über die maximale Verzögerungszeit zulässt.

Das Symbol Window ist genauso wie das dynamische Segment optional und dient ausschließlich zur Überprüfung der Buswächter. Dafür ist das Symbol Windows als kommunikationsfreier Abschnitt des Zyklus definiert worden.
Ein funktionsfähiger Bus Guardian schützt diesen Bereich und unterbindet jegliche Sendeaktivität. Zur Überprüfung, ob dies der Fall ist, wird durch den zugehörigen Kommunikationscontroller der Versuch unternommen, das sogenannte Media Test Symbol (MTS) zu senden. Wenn das gesendete Symbol auf einem der Kommunikationskanäle sichtbar wird, ist eine Fehlfunktion des Bus-Guardians nachgewiesen. Für die Realisierung kann jedem Netzwerkteilnehmer ein Zyklus für die Überprüfung des Bus Guardians zugewiesen werden, sodass über den Zyklus, in dem ein Media Test Symbol empfangen wird, der entsprechende Bus Guardian detektiert werden kann. Somit können, bis auf den zyklusspezifischen Sender des MTS alle Knoten für die Überprüfung herangezogen werden [Raus2008].

Die Network Idle Time ist der letzte Abschnitt jedes Kommunikationszyklus. Durch diesen ebenfalls kommunikationsfreien Bereich des Zyklus ist sichergestellt, dass die Kommunikationscontroller der Netzwerkknoten, die Synchronisation ihrer lokalen Uhren durchführen können (vgl. Kapitel 3.5).

3.5 Frame-Format

Die Nachrichten, die in den Static-Slots beziehungsweise Minislots des statischen und dynamischen Segments von den Teilnehmern eines FlexRay-Netzwerks versendet werden, heißen Frames. Ein Frame setzt sich aus einem Header Segment, einem Payload Segment und einem Trailer Segment zusammen(vgl. Abbildung 10).

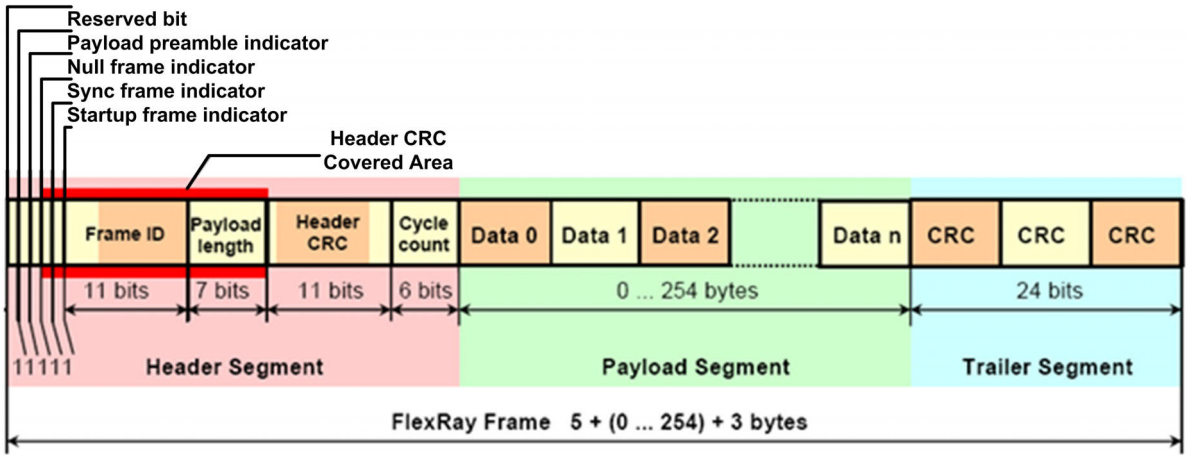

Abbildung 10 – FlexRay Frame Format [Flex2004]

Das Header Segment besteht aus fünf Bytes. Davon sind die folgenden ersten fünf Bits für Steuerinformationen vorgesehen, durch die spezielle Frames gekennzeichnet werden können:

- Das Reserved Bit ist für zukünftige Protokollerweiterungen vorgesehen.
- Mit dem Payload Preamble Indicator können Nachrichten gekennzeichnet werden, dessen Payload Segment spezielle Daten enthält, die durch den Empfänger entsprechend interpretiert werden müssen. Die Interpretation ist außerdem davon abhängig, ob die empfangende Nachricht im dynamischen oder statischen Segment übertragen worden ist. Im dynamischen Segment zeigt der Payload Preamble Indicator an, dass die ersten beiden Bytes der Payload als Message Identifier verwendet werden. Durch diesen, vom Anwender frei wählbaren Message Identifier, unterstützt das FlexRay-Protokoll die Interoperabilität mit CAN-Systemen. Im statischen Segment wird den Empfängern durch den Payload Preamble Indicator angezeigt, dass die Payload ein Network Management Vector enthält, der für den Austausch und die Verarbeitung von Netzwerk-Management-Daten vorgesehen ist. Anhand der Daten im Network Management Vector kann beispielsweise jeder Knoten im Netzwerk entscheiden, ob weiterer Kommunikationsbedarf besteht oder er in einen Standby-Modus wechseln kann [Flex2004] [Raus2008].

- Der Nullframe Indicator wird durch die Kommunikationscontroller der FlexRay-Knoten gesetzt, um den Empfängern einer Nachricht anzuzeigen, dass die Payload des Frames mit Nullen aufgefüllt ist und somit keine brauchbaren Daten enthält. Dazu kann es kommen, wenn der Host des Sende-Knotens dem zugehörigen Kommunikationscontroller die Daten nicht rechtzeitig zur Verfügung stellt. Da das Senden der Frames im statischen Segment für die Synchronisations-Mechanismen und die Startphase des Netzwerks aber unverzichtbar ist, wird in diesem Fall ein vom Kommunikationscontroller selbstständig generierter Nullframe gesendet [Raus2008].
- Der Sync Frame Indicator kennzeichnet einen Frame, der für die systemweite Synchronisation der Kommunikation eingesetzt werden soll. Auf die Realisierung dieses Verfahrens wird in dem folgenden Kapitel 3.6 näher eingegangen.
- Durch den Startup Frame Indicator werden sogenannte Startup Frames angezeigt, die für eine besondere Aufgabe während der Startphase eines FlexRay-Netzwerks vorgesehen sind. Im Kapitel 3.7 wird dies genau erläutert.

Darauf folgen die Frame-ID, die angibt, in welchem Slot eines Zyklus die Nachricht übertragen werden soll und das Feld Payload Length, das die Anzahl der Bytes im Payload Segment angibt. Diese beiden Bereiche, sowie Sync- und Startup Frame Indicator werden durch die darauf folgende Prüfsumme, den sogenannten Header-CRC abgesichert. Durch diese Prüfung können Übertragungsfehler der Elemente, die für die beschriebenen, elementaren FlexRay-Mechanismen vorgesehen sind, festgestellt werden.
Das Ende des Header Segments ist für den Zykluszähler vorgesehen, durch den angezeigt wird, in welchem Zyklus ein Frame übertragen wird [Flex2004].

Im Payload Segment können bis zu 254 Bytes an Nutzdaten untergebracht und von der Anwendung für den eigentlichen Nachrichtenaustausch im Netzwerk verwendet werden. Die Länge der Payload wird dabei in 16-Bit-Worten konfiguriert, sodass sie immer aus einer geraden Anzahl von Bytes besteht [Raus2008].

Das Trailer Segment am Ende jedes Frames beinhaltet einen 24-Bit-CRC, mit dem in der gesamten Nachricht Übertragungsfehler festgestellt werden können.
Die Güte der Fehlererkennung ist von der Länge eines Frames abhängig. So können bis zu sechs fehlerhafte Bits erkannt werden, wenn die Länge mit maximal 248 Bytes konfiguriert ist. Dagegen kann die Fehlererkennung bei größeren Längen nur für vier fehlerhafte Bits garantiert werden [Flex2004].

3.6 Uhrensynchronisation

Für das im FlexRay-Protokoll verwendete zeitgesteuerte Buszugriffsverfahren ist es zwingend erforderlich, dass alle Kommunikationsteilnehmer das gleiche Zeitverständnis haben. Da die Oszillatoren, die auf den Knoten für die Zeitgenerierung zuständig sind, herstellungsbedingt Schwankungen aufweisen, die durch äußere Einflüsse, wie Temperaturschwankungen noch verstärkt werden können, sind Synchronisationsalgorithmen notwendig. Diese Algorithmen ermöglichen es, Abweichungen der lokalen Uhren der Kommunikationscontroller und der Bus Guardians anhand einer globalen Zeitbasis zu korrigieren.

3.6.1 Zeithierarchie

Die globale Zeitbasis, an der die Synchronisation ausgerichtet wird, wurde im FlexRay-Protokoll durch eine Hierarchie von Zeiteinheiten realisiert und ist in Abbildung 11 dargestellt.

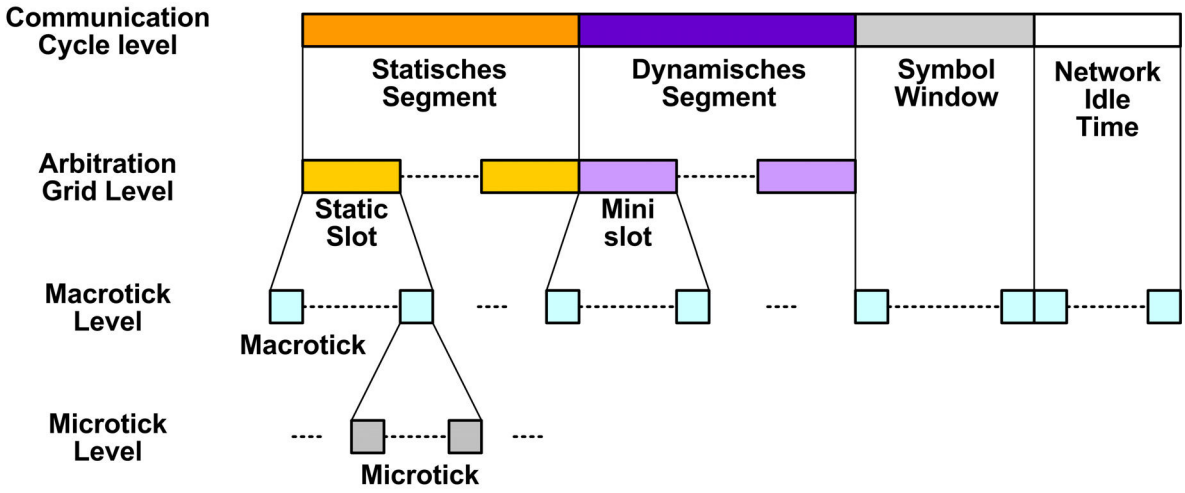

Abbildung 11 – Zeithierarchie im Kommunikationszyklus [Temp2003]

Ein Microtick stellt die kleinste definierte Zeiteinheit dar und kann auf jedem Knoten von unterschiedlicher Länge sein, da diese aus der Taktfrequenz der lokalen Oszillatoren abgeleitet wird. Während der Planungsphase und der FlexRay-Konfiguration kann die Länge eines Microticks mit einem Wert von 12,5 bis zu 100 Nanosekunden festgelegt werden.
Aus den Microticks wird die nächst größere Zeiteinheit, die Macroticks gebildet. Macroticks haben die Eigenschaft, dass sie in allen Knoten gleich groß sind und stellen somit die globale Zeiteinheit für das gesamte Netzwerk dar.
Die Anzahl Microticks, die einen Macrotick formen, kann aber durch die unterschiedlichen

Längen der Microticks von Knoten zu Knoten variieren. Mit Hilfe dieser Relation erfolgt die Synchronisation der Netzwerkteilnehmer, indem jeder Knoten die Anzahl von Microticks, die einen Macrotick bilden, in bestimmten Abständen anpasst und damit die Abweichung seiner lokalen Uhr korrigieren kann. Das FlexRay-Protokoll sieht bei der Konfiguration eines Macroticks durch den System-Designer Längen zwischen einer und sechs Mikrosekunden vor, wobei ein Macrotick außerdem aus mindestens 40 Microticks bestehen muss. Durch diese zusätzliche Einschränkung ist gewährleistet, dass die beschriebene Verlängerung beziehungsweise Verkürzung eines Macroticks während der Uhrensynchronisation durchgeführt werden kann. Des Weiteren ist die maximale Länge eines Macroticks auf 240 Microticks begrenzt, um die Implementierung in einem 8-Bit-Register zu ermöglichen.
Eine Konfiguration eines FlexRay-Netzwerks, bei der eine Mikrosekunde als Länge eines Macroticks gewählt worden ist, könnte beispielsweise ergeben, dass der Macrotick auf dem Knoten A aus 40 und auf dem Knoten B aus 80 Microticks gebildet wird, da die Länge eines Microticks auf Knoten A mit 25 Nanosekunden und auf dem Knoten B mit 12,5 Nanosekunden konfiguriert ist.
Auf der Ebene der Buszuteilung, dem sogenannten Arbitration Grid Level, wird die Größe der Static-Slots im statischen Segment und die Größe der Minislots im dynamischen Segment auf Basis der Macroticks im Zuge der FlexRay-Konfiguration definiert.
Auf der höchsten Ebene der Hierarchie ist der Kommunikationszyklus angesiedelt, der damit die größte Zeiteinheit darstellt und dessen Länge aus einer feststehenden Anzahl von Macroticks gebildet wird [Raus2008] [Geva2005].

3.6.2 Korrekturmaßnahmen

Zur Korrektur der zeitlichen Abweichungen der lokalen Uhren kommen zwei verschiedene Methoden, die Offsetkorrektur und die Steigungskorrektur, zum Einsatz.

Offsetkorrektur
Die erste, in Abbildung 12 dargestellte Methode ist die Offsetkorrektur. Die Idealzeit ist hier durch die Abszisse und die zu synchronisierenden Uhren von zwei Kommunikationsteilnehmern durch die farbigen Kennlinien visualisiert.
Es ist ersichtlich, dass durch die unterschiedlichen Frequenzen der Uhren, eine konstant wachsende Abweichung gegenüber der Idealzeit entsteht und es daher erforderlich ist, in regelmäßigen Abständen eine Korrektur der Uhren vorzunehmen.
Bei der Offsetkorrektur wird dazu der Wert der Abweichung ermittelt, um damit die von der Uhr angezeigte Zeit zu ändern [Raus2008].
Im FlexRay-Protokoll ist dafür die Network Idle Time vorgesehen, in der der Kommunikationszyklus und durch das Einfügen beziehungsweise Entfernen von Microticks in seiner zeitli-

chen Länge, der gemessenen Abweichung entsprechend angepasst wird.

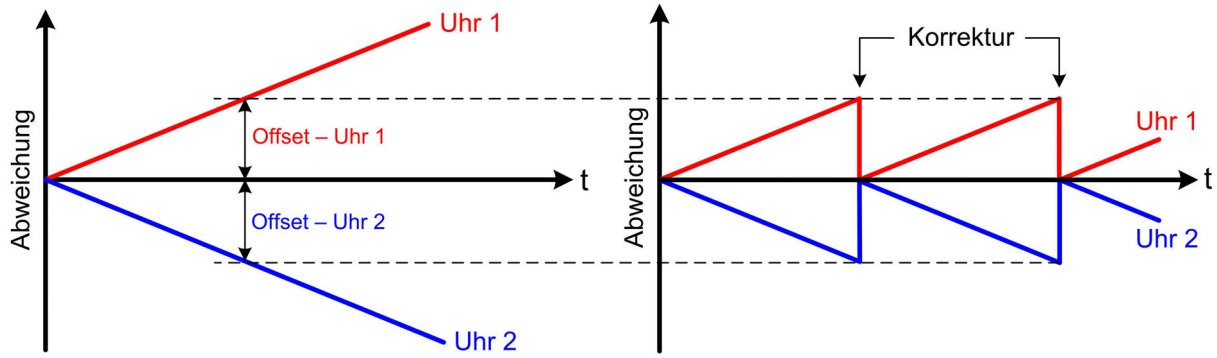

Abbildung 12 – Offsetkorrektur [Raus2008]

Steigungskorrektur

Da die beschriebene Offsetkorrektur die Ursache der Abweichungen nicht korrigiert und die korrigierte Uhr nach einiger Zeit wieder noch der Idealzeit abweicht, wird die Steigungskorrektur als zweite Methode zur Synchronisation eingesetzt.

Durch die Anpassung des Verhältnisses von Microticks pro Macrotick ist die Möglichkeit gegeben, die Frequenzen der Uhren anzugleichen (vgl. Abbildung 13). So ist gewährleistet, dass die Uhren über einen längeren Zeitraum synchronisiert bleiben und die Abstände zwischen den Offsetkorrekturen vergrößert werden können.

Die Bestimmung der, für die Offset- und Steigungskorrektur benötigten Werte wird im folgenden Abschnitt erläutert.

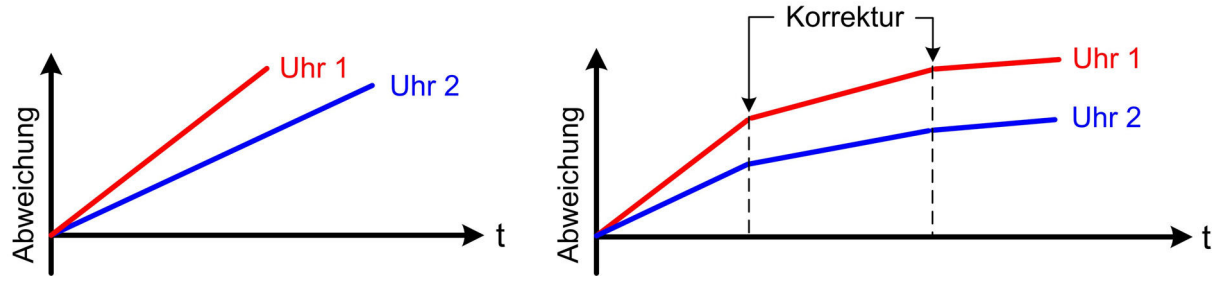

Abbildung 13 – Steigungskorrektur [Raus2008]

3.6.3 Bestimmung der Korrekturwerte

Für die Bestimmung der Korrekturwerte sind Messungen der Uhrenabweichungen zwischen den Knoten erforderlich. Diese Messungen werden im statischen Segment des Kommunikationszyklus durchgeführt, da hier ein fester Zeitplan für das Senden der Nachrichten vorliegt. Jedem Knoten ist demnach der Zeitpunkt bekannt, zu dem eine Nachricht ankommen müsste und kann so die Abweichung zwischen erwarteter und tatsächlicher Ankunftszeit bestimmen. Prinzipiell können die Netzwerkknoten diese Messung bei jeder Nachricht, die im statischen Segment empfangenden wird, durchführen. Im FlexRay-Protokoll ist für die Bestimmung der Abweichung und die darauf folgende Offsetkorrektur ein spezieller Nachrichtentyp festgelegt worden, die Synchronisationsnachricht. So ist dem System-Designer eines FlexRay-Netzwerks die Möglichkeit gegeben, bestimmte Kommunikationsteilnehmer auszuwählen, die für den Versand der Synchronisationsnachrichten zuständig sein sollen und als sogenannte Sync-Knoten konfiguriert werden. Durch dieses Entwurfskriterium, das in Kapitel 3.9 näher beschrieben wird, kann dem Fehlerfall der sogenannten Cliquenbildung entgegengewirkt werden.
Die gesendeten Synchronisationsnachrichten, die auch als Sync-Frames bezeichnet werden, können bei den Empfängern durch den gesetzten Sync Frame Indicator im Header Segment als solche identifiziert werden. Jeder Knoten bestimmt die Differenz aus der Ankunftszeit der Sync-Frames und dem erwarteten Zeitpunkt, speichert diese Abweichungen und verwendet sie am Ende jedes zweiten Zyklus für die Berechnung der Korrekturwerte und die anschließende Offsetkorrektur.
Dagegen ist es für die Steigungskorrektur erforderlich, dass zwei Messungen in aufeinander folgenden Kommunikationszyklen durchgeführt werden (vgl. Abbildung 14).

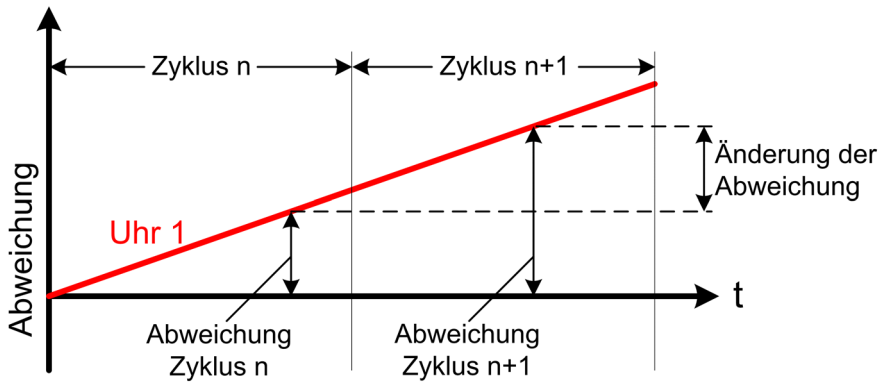

Abbildung 14 – Bestimmung der Abweichungen [Raus2008]

Die Differenz aus diesen beiden Messungen ist die Steigungsdifferenz, die anzeigt, um wie viel sich die lokale Uhr pro Zyklus von der Idealzeit entfernt, wenn keine Korrektur vorge-

nommen wird. Außerdem entfällt aufgrund der Differenzbildung, die in beiden Messwerten enthaltende Laufzeit der Nachrichten und führt so zu einer hohen Qualität der Steigungskorrektur [Raus2008].

Die eigentliche Berechnung wird mit einem fehlertoleranten Mittelpunktalgorithmus durchgeführt. Bei diesem Algorithmus werden die Messwerte der Größe nach geordnet, und in einer Liste gespeichert. Bevor der Mittelwert gebildet wird, werden die größten und kleinsten Werte aus der Liste verworfen. So kann sichergestellt werden, dass fehlerbehaftete Messwerte keinen Einfluss auf die dann folgende Synchronisation haben.
In Tabelle 1 ist das Verfahren für die Steigungskorrektur mit exemplarischen Werten dargestellt. Bei der Anwendung des Algorithmus für die Offsetkorrektur entfällt die Differenzbildung, da hierbei lediglich die Werte eines Zyklus benötigt werden.

Sync-Frames	Abweichung Zyklus n [Microticks]	Abweichung Zyklus n+1 [Microticks]	Differenz [Microticks]	Ordnen [Microticks]	Berechnung [Microticks]
Static-Slot 1	2	7	5	~~9~~	$\frac{5+3}{2}=4$
Static-Slot 2	12	15	3	5	
Static-Slot 4	3	12	9	3	
Static-Slot 6	22	21	-1	~~-1~~	

Tabelle 1 – Berechnung der Steigungskorrekturwerte

Außerdem gibt es zur zusätzlichen Absicherung die Möglichkeit, bei der Konfiguration des FlexRay-Netzwerks die maximalen Grenzen der Korrekturwerte vorzugeben. Sollten diese Grenzen bei der Berechnung verletzt werden, wird die aktive Kommunikation dieses Knotens mit einer Fehlermeldung an den Host eingestellt.

3.7 Starten des Kommunikationsnetzwerks

Während des normalen Betriebs ist durch das Senden der Sync-Frames die Uhrensynchronisation und damit das gemeinsame Verständnis der Zeit für alle Knoten sichergestellt.
Beim Starten eines Kommunikationsnetzwerks ist diese globale Zeitbasis, auf der die Kommunikation beruht, jedoch noch nicht etabliert. Daher muss der Startvorgang eines FlexRay-Systems als Sonderfall betrachtet werden. In Abbildung 15 ist ein solches Szenario exemplarisch dargestellt.

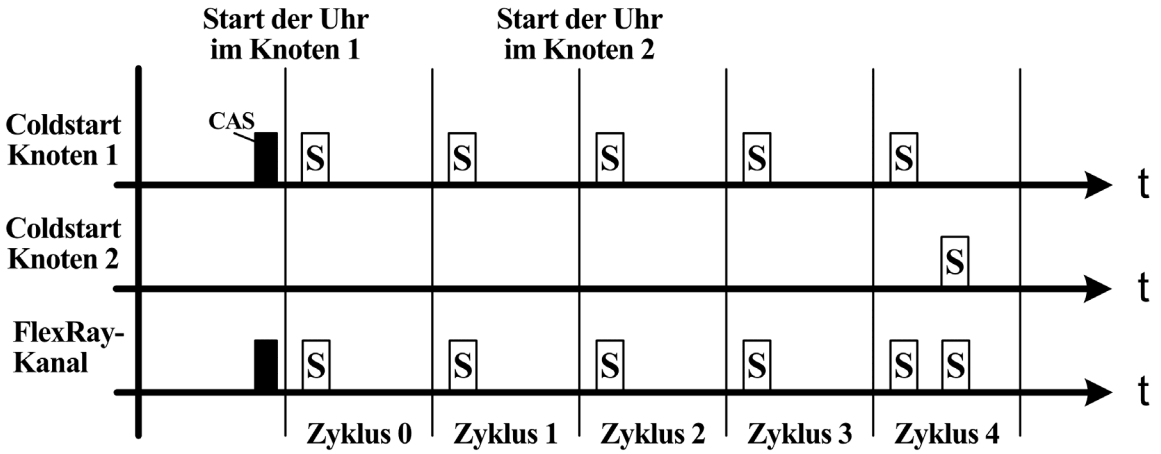

Abbildung 15 – Startup-Phase eines FlexRay-Netzwerks

Im FlexRay-Protokoll sind für diese Phase die sogenannten Coldstarter vorgesehen.
Wird ein als Coldstarter konfigurierter Knoten gestartet, prüft dieser für die Zeit von zwei Kommunikationszyklen, ob bereits andere Netzwerkteilnehmer mit dem Senden begonnen haben. Wenn in diesem Zeitraum keine Aktivität auf den Übertragungskanälen festgestellt wurde, wird das sogenannte Collision Avoidance Symbol (CAS) gesendet, das einer genau definierten Anzahl von Null-Bits entspricht.
Der Sender dieses Symbols wird zum sogenannten Leading Coldstarter und verhindert damit, dass andere Knoten ebenfalls mit dem Senden beginnen. Die übrigen als Coldstarter konfigurierten Netzwerkteilnehmer, die das Collision Avoidance Symbol detektieren, werden als Following Coldstarter bezeichnet.
Der Leading Coldstarter startet seine interne Uhr und sendet gemäß dem festgelegten TDMA-Plan, in dem ihm zugeordneten Static-Slots. Die gesendeten Nachrichten werden dabei durch den Startup Frame Indicator im Header Segment als Startup-Frames gekennzeichnet.
Die Startup-Frames übernehmen während der Startphase die Rolle der Synchronisationsnachrichten und ermöglichen den Following Coldstartern so die Durchführung der Uhrensynchronisation. Für die Synchronisation während des Startvorgangs werden vier Startup-Frames in

aufeinanderfolgenden Kommunikationszyklen benötigt. Nach den ersten beiden Zyklen können die Following Coldstarter ihre lokalen Uhren starten, mit der empfangenden Zyklusnummer initialisieren und dann durch das in Kapitel 3.6 beschriebene Verfahren synchronisieren. Durch den Empfang der nächsten beiden Startup-Frames wird geprüft, ob die daraus berechneten Korrekturwerte innerhalb der konfigurierten Grenzen liegen und die Synchronisation erfolgreich war. Ist das der Fall, können die Following Coldstarter ins Netzwerk integriert werden und ebenfalls mit dem Senden in den, ihnen zugeordneten Slots beginnen.
Sobald der Leading Coldstarter den Sendevorgang anderer Knoten feststellt, beendet er die Startup-Phase und wechselt in den normalen Betriebsmodus.
Knoten, die nicht als Coldstarter konfiguriert worden sind, warten mit ihrer Synchronisation bis sie Frames von zwei verschiedenen Coldstartern empfangen haben und integrieren sich wie die Following Coldstarter nach vier Kommunikationszyklen [Reif2007] [Raus2008].

3.8 FlexRay–Topologien

Neben der Möglichkeit in einem FlexRay-Netzwerk Fehlertoleranz durch die redundante Nutzung der beiden zur Verfügung stehenden Übertragungskanäle zu erreichen und der Absicherung des TDMA-Plans der Netzwerkknoten durch Verwendung der Bus Guardians, kann auch die Topologie des Systems zur Erhöhung der Verlässlichkeit beitragen. FlexRay unterstützt bei der physikalischen Anordnung der Knoten die Bus- und Sterntopologie, die auch als Mischform eingesetzt werden können [Raus2008].
Abbildung 15 zeigt eine einkanalige Bustopologie und einen exemplarischen Kurzschluss der Busverbindung. Es ist ersichtlich, dass ein solches Fehlerszenario den kompletten Ausfall der Kommunikation zur Folge hätte.
Für sicherheitskritische Anwendungsbereiche müsste bei einer Worst-Case-Analyse in Betracht gezogen werden, dass der Kurzschluss auch innerhalb eines Knotens auftreten kann und es so bei einer redundanten, zweikanaligen Topologie ebenfalls zu einem Totalausfall der Kommunikation kommen könnte [Elen2004].

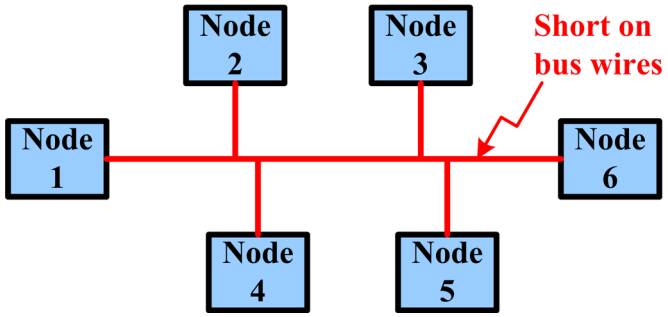

Abbildung 16 – Einkanalige Bustopologie [Mill2005]

Zusätzliche Sicherheit ist durch die in Abbildung 16 dargestellte Sterntopologie erreichbar. Bei dieser Art der Topologie wird ein aktiver Sternkoppler eingesetzt, der die eingehenden elektrischen Signale verstärkt und verteilt.

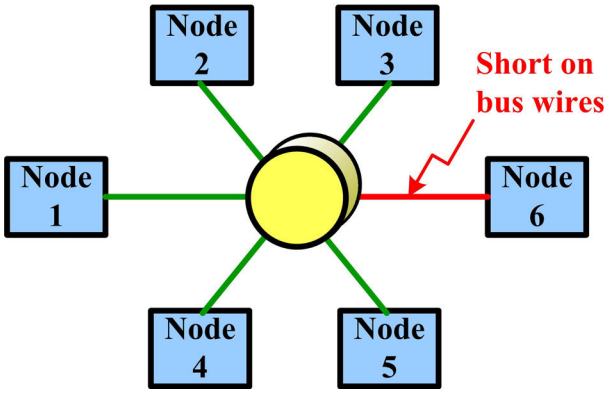

Abbildung 17 – Sterntopologie [Mill2005]

Die angestrebte Erhöhung der Verfügbarkeit mittels Einsatz eines aktiven Sternkopplers entsteht durch die Eigenschaft, dass er blockierte Zweige und auch fehlerhafte Nachrichten erkennen kann. Mit der Erkennung eines solchen Falls wird der betroffene Zweig deaktiviert und die Fehlerquelle isoliert. Überdies wäre die Erweiterung der abgebildeten Sterntopologie zu einem Design mit zwei Kanälen denkbar, um den Sternkoppler als “single point of failure“ zu vermeiden.

Prinzipiell sind die Sterntopologien eines FlexRay-Systems auch mit passiven Sternkopplern realisierbar. Da ein passiver Sternkoppler im Wesentlichen nur die elektrische Verbindung zwischen den angeschlossenen Netzwerkknoten herstellt, wird damit für sicherheitsrelevante Anwendungen kein Vorteil in bezug auf die Verlässlichkeit erreicht.

3.9 Cliquenbildung

In Kommunikationssystemen, die wie FlexRay mit einem dezentral gesteuerten Synchronisationsverfahren arbeiten, kann es zur Entstehung von Cliquen kommen.
Eine Clique stellt eine Gruppe mehrerer Kommunikationsteilnehmern dar, die miteinander synchronisiert sind und Nachrichten austauschen können. Außerhalb dieser Gruppe existieren weitere Knoten, die ebenfalls miteinander synchronisiert sind. Aber im Verhältnis zur ersten Gruppe hat sich bei diesen Knoten ein verschobenes Zeitverständnis etabliert, sodass die Kommunikationszyklen der beiden Gruppen zu unterschiedlichen Zeitpunkten beginnen. Ein solcher Fehlerfall hat die Folge, dass der exklusive Buszugriff nicht mehr gewährleistet ist und es zur Kollision von Frames kommen kann [Raus2008].
Zur Verdeutlichung der Cliquenbildung ist in Abbildung 18 ein FlexRay-Netzwerk dargestellt, das mit einer Mischform aus Stern- und Bustopologie aufgebaut ist. Hierbei kann beispielsweise der Fall eintreten, dass der dargestellte Sternkoppler während der Startphase des Netzwerks noch nicht betriebsbereit ist und kein Nachrichtenaustausch zwischen dem linken und dem rechten Zweig stattfindet. Somit können sich nur die Knoten im jeweiligen Zweig aufeinander synchronisieren und bilden damit je eine Clique. Der Nachrichtenaustausch zwischen den Cliquen ist anschließend auch mit einem betriebsbereiten Sternkoppler nicht mehr möglich, da kein gemeinsames Zeitverständnis etabliert werden konnte.

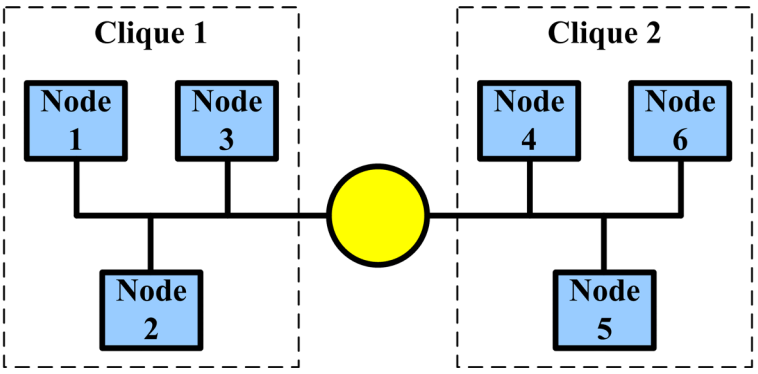

Abbildung 18 – Cliquenbildung in einem FlexRay-Netzwerk

Um die Verlässlichkeit eines Systems sicherzustellen, ist es erforderlich, diesen Fehlerfall zu berücksichtigen und den Entwurf des Kommunikationsnetzwerks so zu planen, dass die Cliquenbildung vermieden wird. Ein Entwurfskriterium, das die Entstehung von Cliquen entscheidend beeinflusst, ist die Anzahl der konfigurierten Sync-Knoten.
Eine Clique, wie auch ein fehlerfreies Netzwerk, benötigt mindestens zwei Sync-Knoten, damit die beschriebene Korrektur der lokalen Uhren durchgeführt werden kann. Daraus ergibt sich, dass nur bei einer Netzwerkkonfiguration mit mindestens vier Sync-Knoten die Möglichkeit der Cliquenbildung gegeben ist. Die Konfiguration eines Netzwerks mit maximal drei

Sync-Knoten ist daher die einfachste Maßnahme, um die Entstehung von Cliquen zu verhindern, aber mit der Beeinträchtigung verbunden, dass höchstens einer der drei Knoten ausfallen dürfte, da ansonsten keine Synchronisation mehr stattfinden kann und die gesamte Kommunikation zum Erliegen käme.
Des Weiteren liegt im Fall der Cliquenbildung auch immer ein Defekt der physikalischen Verbindung vor, der dazu führt, dass bei der Berechnung der Korrekturwerte nicht alle Synchronisationsnachrichten einbezogen werden, sondern nur die der jeweiligen Gruppe.
Die Wahrscheinlichkeit eines solchen Fehlerszenarios, das zum Beispiel beim Ausfall eines Sternkopplers eintritt, kann durch die Verwendung beider Kommunikationskanäle und redundante Auslegung der zugehörigen Komponenten verringert werden.
Die letztendlichen Konfigurationsentscheidungen sind von der konkreten Anwendung und dem zugrunde liegenden Fehlermodell abhängig, wobei stets ein Kompromiss zwischen Verfügbarkeit und Cliquenvermeidung erforderlich ist [Raus2008].

4. Design und Realisierung des Gesamtsystems

Der Entwurf der zeitgesteuerten Architektur eines verteilten Echtzeitsystems kann durch zwei aufeinander aufbauende Entwicklungsschritte systematisiert werden.
Ein Konzept, das unter anderem in der Automobilindustrie zum Tragen kommt, ist der zweistufige Entwicklungsansatz, wobei so die unabhängige und parallele Entwicklung der Vielzahl von Komponenten durch mehrere Zulieferer ermöglicht werden kann.
In Abbildung 19 ist dieser Ansatz in einem V-Modells dargestellt und zeigt die Trennung zwischen dem sogenannten Cluster-Design und dem Komponenten-Design. Dabei verdeutlicht die linke, absteigende Seite die erforderlichen Definitions- und Designphasen und die rechte, aufsteigende Seite die Test- und Integrationsphasen [Tech2007].
Im ersten Schritt wird das Cluster-Design entworfen, das die globale Planung bezüglich des Gesamtsystems beinhaltet. Dabei wird aus den gegebenen Systemanforderungen die zu realisierende Funktion des Gesamtssystem abgeleitet und die Verteilung dieser Funktion auf mehrere Subsysteme vorgenommen. Des Weiteren werden durch den System-Designer die Sendezeitpunkte und die benötigte Bandbreite der einzelnen Subsysteme bestimmt und im globalen Ablaufplan festgehalten, der damit die Koordination des Buszugriffs und die für das Echtzeitverhalten erforderliche, deterministische Kommunikation gewährleistet [Weic2005].
Da Fehler im Design oder unvollständig spezifizierte Anforderungen unter Umständen erst in einem späten Entwicklungsschritt erkannt werden können, ist in dieser Phase eine methodische Vorgehensweise zwingend erforderlich.

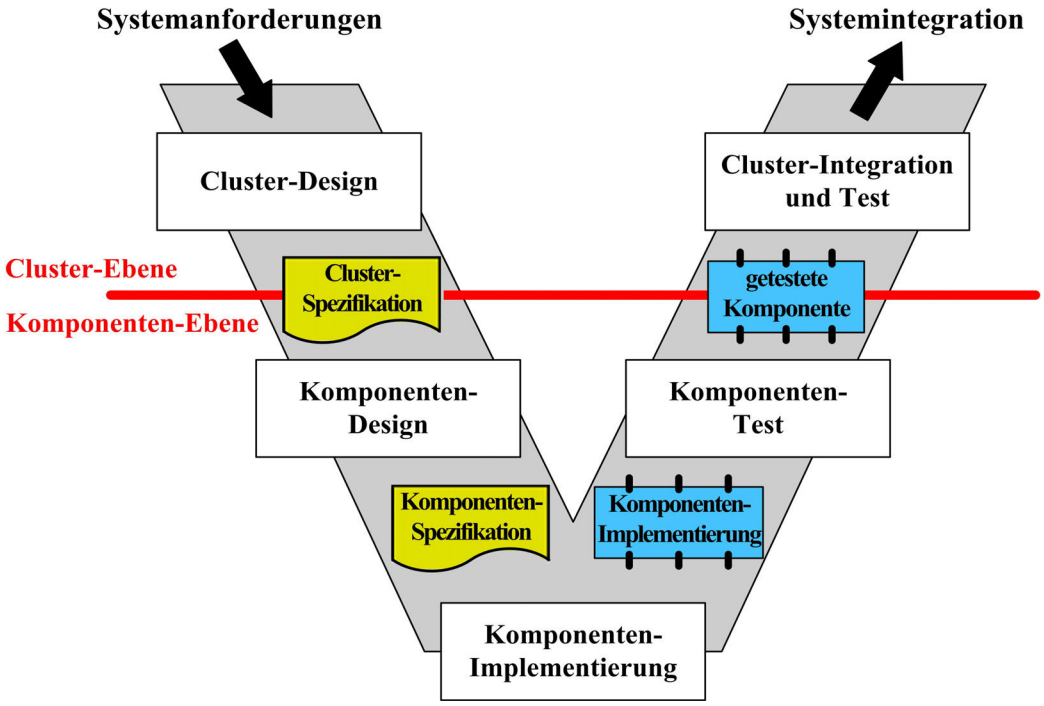

Abbildung 19 - Die Phasen des zweistufigen Entwicklungsansatzes im V-Modell [Weic2005]

Das mit dem ersten Entwicklungsschritt entworfene Design wird in einer Cluster-Spezifikation niedergelegt und stellt die Basis für das darauffolgende Komponenten-Design dar.
Die Reihenfolge dieser Arbeitsschritte ist auch bei Projekten unumgänglich, die weniger Komplexität aufweisen als die eingangs erwähnten Entwicklungen der Automobilbranche, da für das Komponenten-Design einer zeitgesteuerten Architektur die Wechselwirkungen mit anderen Komponenten bekannt und vorab geplant sein müssen. Das Komponenten-Design umfasst die Spezifikation der benötigten Prozesse, die auch als Tasks bezeichnet werden und durch deren Implementierung das gewünschte Verhalten der jeweiligen Komponente realisiert wird. Außerdem müssen in diesem Entwicklungsschritt die Ausführungszeiten der Tasks geplant werden, um den Nachrichtenaustausch gemäß des Zeitplans der Cluster-Spezifikation gewährleisten zu können.
Nachdem alle erforderlichen Komponenten spezifiziert, implementiert und anschließend getestet worden sind, erfolgt die Systemintegration und letztendlich die Zusammenführung zum vollständigen System.
In diesem Kapitel wird der erste Schritt des beschriebenen, zweistufigen Entwicklungsansatzes durchgeführt, um ein verteiltes Echtzeitsystem auf FlexRay-Basis zu realisieren. Zunächst werden die Systemanforderungen spezifiziert, die Entwurfsmöglichkeiten bei der Definition der Subsysteme dargestellt und die Erstellung des globalen Ablaufplans durchgeführt.

4.1 Systemanforderungen

Die Anforderungen, die an das zu entwickelnde System gestellt werden, beruhen auf den Bachelorarbeiten von Haase und Schuckert [Haas2007] [Schu2007]. In diesen vorangegangenen Arbeiten ist ein verteiltes Echtzeitsystem auf TTCAN-Basis entworfen worden, das für Telemetrie- und Steuerungsaufgaben in einem Formula Student Rennwagen eingesetzt wird. Das Hauptaugenmerk der vorliegenden Arbeit liegt auf der Verbesserung der Fehlertoleranz und Zuverlässigkeit der zeitgesteuerten Architektur des gegebenen TTCAN-Netzwerks durch die Migration zu einem FlexRay-Kommunikationssystem.
Dies wird durch die in Kapitel 3 beschriebenen Eigenschaften des FlexRay-Protokoll erreicht, mit denen die Möglichkeit gegeben ist, ein verlässliches Echtzeitsystem zu entwerfen, das den Entwicklungen der Automobilindustrie folgt und zukünftig auch für sicherheitsrelevante Steuerungsaufgaben einsetzbar sein wird.

4.1.1 Funktionale Anforderungen

Die für die durchzuführende Migration zu spezifizierenden Systemanforderungen sind zum einen die realisierten Funktionen des TTCAN-Systems, die in der folgenden Auflistung dargestellt sind:

- Erfassung des Lenkwinkels.
- Erfassung der Raddrehzahlen
- Erfassung der Daten der Motorsteuerung
- Aufzeichnung aller anfallenden Daten.
- Liveübertragung aller Daten zu einem Kontrollstand

Dabei ist die Datenübertragung so auszulegen, dass eine höchstmögliche Auslastung des Bussystems erreicht wird.
Zum anderen sind in Tabelle 2 die Sensoren der Motorsteuerung im Einzelnen aufgeschlüsselt und die minimal benötigten Frequenzen aufgeführt, mit denen die zugehörigen Sensorwerte ausgelesen und übertragen werden müssen.

Typ	Sensor / Wert	Sollfrequenz [Wert/s]
Motorsteuerung	EngineRoundCounter	5
	MAP	5
	Phase1	5
	TAir	5
	TPS	25
	RPM	25
	Klambda1	25
	Inj1, Inj2, Inj3, Inj4	25
	ValvPosition	25
	Spark1, Spark2, Spark3, Spark4	25
	Sidestand	25
	Lambda	25
	VBatt	5
	DFarf	25
	TEngine	5
	LambdaTarget	5
	Speed	2
	Tipover	25
	Gear	2
	Active Block	1
	BAP	5
Sensoren	Raddrehzahl vorne links / rechts	25
	Raddrehzahl hinten links / rechts	25
	Lenkwinkelsensor	25

Tabelle 2 - Sollfrequenzen der Sensoren und Steuergeräte[Haas2007]

Aktuelle und zukünftige Entwicklungen
Außerdem müssen die aktuellen Entwicklungen, die während der Anfertigung der vorliegenden Arbeit entstanden sind, bei der Erhebung der Systemanforderungen berücksichtigt werden. Dazu zählt eine Antischlupfregelung, die im gegenwärtigen Entwurf eines Formula Student Rennwagens an der Hochschule für Angewandte Wissenschaften Hamburg realisiert worden ist und das erste von mehreren Fahrerassistenzsystemen darstellt, die für nachfolgende Modelle des Rennwagens geplant sind. Ein weiteres, bereits geplantes Fahrerassistenzsystem sieht die elektronische Stabilisierung des Fahrzeugzeugs vor.

4.1.2 Nichtfunktionale Anforderungen

Verlässlichkeit
Die nichtfunktionalen Anforderungen an das System beziehen sich zum einen auf die sicherheitsrelevanten Steuerungen mit denen die Fahrerassistenzsysteme realisiert werden. Hierfür ist ein verlässliches System erforderlich, das die damit verbundenen Eigenschaften, wie Fehlertoleranz, Verfügbarkeit und Zuverlässigkeit gewährleisten muss.

Leistung und Effizienz
Zum anderen ist das System für die telemetrische Datenerfassung so zu entwickeln, dass alle anfallenden Daten verarbeitet werden können. Sowohl für die Speicherung als auch für die direkte Visualisierung der Daten ist bei der Entwicklung das Leistungsverhalten des Systems zu berücksichtigen.

Wartbarkeit und Änderbarkeit
Des Weiteren muss der Entwurf so ausgelegt sein, dass eine Einarbeitung möglichst einfach erfolgen kann, um die Weiterentwicklung des Systems in nachfolgenden Projekten zu gewährleisten.

Aus den erhobenen Anforderungen kann unter anderem die erforderliche Bandbreite des Systems abgeleitet werden und daraus die für die Cluster-Spezifikation benötigten Sendezeitpunkte festgelegt werden. Für den Versuchsaufbau im Labor werden die aufgelisteten Sensorwerte der Motorsteuerung durch ein Software-Modul erzeugt und somit die für den realen Einsatz benötigte Last bei der Datenübertragung nachgebildet.
Ein weiterer entscheidender Punkt bei dem Entwurf des Cluster-Design stellt die beschriebene Aufteilung der Funktionalität in Subsysteme dar, die in diesem Fall bereits durch das entwickelte System von Haase und Schuckert gegeben ist. Da für die vorliegende Arbeit lediglich eine begrenzte Anzahl von Hardware-Komponenten zur Verfügung steht und die verfügbare Entwicklungszeit für die Adaption aller Komponenten nicht ausreichend ist, wird die Aufteilung ebenfalls an die Laborumgebung angepasst und im folgenden die dafür in Frage kommenden Möglichkeiten erläutert.

4.2 Planung der Subsysteme

Durch die gegebene Spezifikation der Systemanforderungen kann die für das Cluster-Design vorgesehene Aufteilung der Funktionen auf die Subsysteme erfolgen. In Abbildung 20 ist ein möglicher Versuchsaufbau des zu entwerfenden FlexRay-Kommunikationssystems für die Nachbildung der realen Verhältnisse im Labor dargestellt.

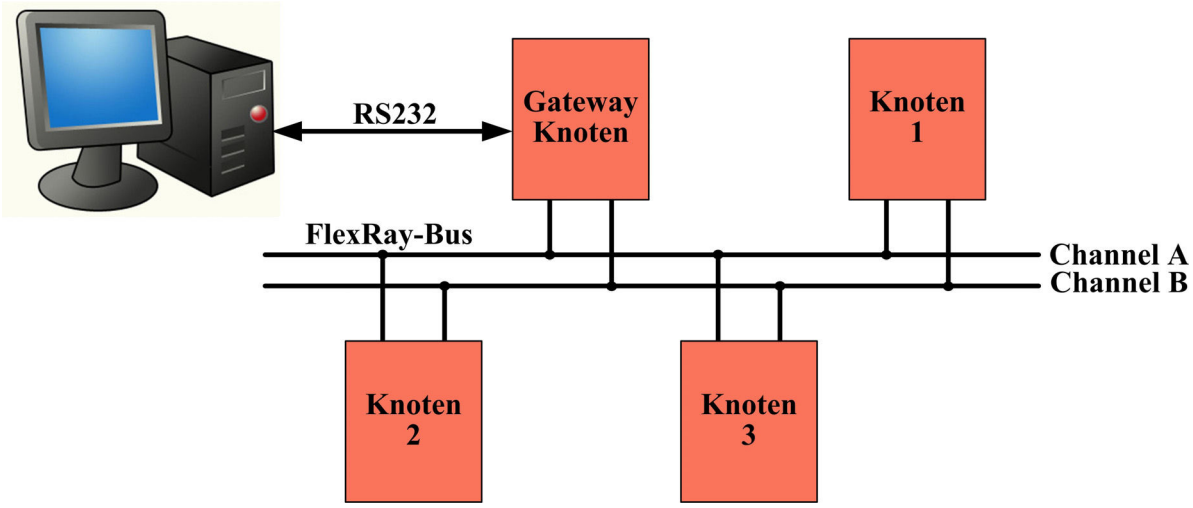

Abbildung 20 – Versuchsaufbau des FlexRay-Systems im Labor

Ein Netzwerkknoten wird als Gateway konfiguriert, um die Verbindung aus dem FlexRay-Netzwerk zu einem Rechner herzustellen, der für die Aufzeichnung aller anfallenden Daten eingesetzt wird. Die Verbindung, die im TTCAN-System durch ein WLAN-Modul realisiert ist, wird aufgrund der für das FlexRay-System verfügbaren Hardware über die serielle Schnittstelle RS232 umgesetzt. So kann die Liveübertragung der Daten im Labor simuliert und mit Hilfe einer Benutzeroberfläche visualisiert werden.
Für die Aufteilung der Funktionen auf die übrigen Knoten ist neben den funktionalen Anforderungen auch die räumliche Anordnung der Sensoren im System entscheidend. Zum Beispiel sind die Sensoren, die für die Ermittlung der Radumlaufgeschwindigkeiten eingesetzt werden, an den Rädern der Vorderachse und der Hinterachse des Formula Student Rennwagens angebracht, sodass für die Achsen je ein Knoten zu verwenden ist. Demnach ist der dritte verfügbare Knoten für die Erzeugung und Übertragung aller Sensorwerte der Motorsteuerung und des Lenkwinkelsensors einzusetzen.
Da bei einer solchen Konstellation die beiden für die Ermittlung der Raddrehzahlen verwendeten Module identisches Verhalten aufweisen würden und sich nur durch verschiedene Sendezeitpunkte unterscheiden, wird für den Laboraufbau die räumliche Anordnung der Sensoren vernachlässigt und eine zweite Variante für die Aufteilung der Subsysteme gewählt, die sich wie folgt darstellt:

- Ein Modul ermittelt und überträgt die Umlaufgeschwindigkeiten der Räder an der Vorder- und Hinterachse.
- Das zweite Modul generiert einen Teil der Sensorwerte der Motorsteuerung und nimmt die Übertragung der Daten vor.
- Mit dem dritten Knoten werden die Werte des Lenkwinkelsensors generiert, sowie die übrigen Sensorwerte der Motorsteuerung.

4.3 Kommunikationsplanung

Durch die spezifizierten Systemanforderungen und die Definition der Subsysteme mit der zugehörigen Abbildung auf bestimmte Netzwerkknoten sind die grundlegenden Voraussetzungen für die Kommunikationsplanung gegeben. Da der Einsatz des zu entwickelnden FlexRay-Systems auf die Datenerfassung aller anfallenden Sensorwerte ausgelegt ist, wird ausschließlich der als Gateway konfigurierte Netzwerkknoten für den Empfang und die Weiterverarbeitung der Daten eingesetzt.
Die daraus hervorgegangenen Kommunikationsbeziehungen zwischen den Knoten sind in Tabelle 3 dargestellt.

Sender	Empfänger	Sensorwerte / Daten der Nachricht
Knoten 1	Gateway-Knoten	Raddrehzahl vorne/hinten
Knoten 2	Gateway-Knoten	TPS
		RPM
		Klambda1
		Inj3, Inj4
		ValvPosition
		Spark3, Spark4
		Sidestand
		Lambda
		TEngine
		Speed
		Active Block, BAP
Knoten 3	Gateway-Knoten	EngineRoundCounter
		MAP
		Phase1
		TAir
		Inj1, Inj2
		Spark1, Spark2
		VBatt
		DFarf
		LambdaTarget
		Tipover
		Gear
		Lenkwinkelsensor
Gateway-Knoten	Knoten 1, 2 & 3	Konfigurationsdaten

Tabelle 3 – Einteilung der Nachrichten für die Kommunikationsplanung

Es wird verdeutlicht, welche Nachrichten von den jeweiligen Knoten übertragen werden und welche Knoten für den Empfang dieser zugeteilt sind. Außerdem ist ersichtlich, welche Sensorwerte mit den jeweiligen Nachrichten übertragen werden.
Die Ausnahme dazu stellt die vom Gateway-Knoten gesendete Nachricht dar, die für die Übertragung von Konfigurationsdaten eingesetzt wird. Dadurch ist die Möglichkeit gegeben, im laufenden Betrieb konfigurierenden Einfluss auf das Verhalten der Netzwerkknoten zu nehmen.
Die benötigten Konfigurationsdaten werden am Rechner, der für die Datenaufzeichnung eingesetzt wird, über die Benutzeroberfläche eingelesen und über die serielle Schnittstelle an den

Gateway-Knoten übertragen (vgl. Abbildung 20). Dieser prüft in regelmäßigen Abständen, ob neue Daten im Empfangspuffer der Schnittstelle vorliegen und leitet sie gegebenenfalls über den FlexRay-Bus an alle Netzwerkteilnehmer weiter. Anschließend erfolgt die Interpretation der Konfigurationsdaten um festzustellen, welche Änderungen vom Anwender vorgesehen und auf welchem Knoten diese durchzuführen sind.

4.4 Entwurf des TDMA-Ablaufplans

Nach der Festlegung der logischen Abhängigkeiten zwischen den Knoten folgt die Definition der zeitlichen Abfolge des Nachrichtenaustauschs in Form des TDMA-Ablaufplans. Dies beinhaltet die Berechnung des Zyklus, mit dem die Nachrichtenübertragung wiederholt werden muss, um die geforderten Sollfrequenzen einzuhalten und das zu realisierende Echtzeitverhalten zu gewährleisten. Des Weiteren müssen die Zeitschlitze berechnet werden, aus denen ein Zyklus gebildet wird und mit denen den Kommunikationsteilnehmern die erforderliche Bandbreite für die kollisionsfreie Nachrichtenübertragung reserviert wird.
Für die Berechnung der Zykluslänge können die in den Systemanforderungen festgelegten Sollfrequenzen zu Grunde gelegt werden. Die Anforderungen dieser Arbeit sehen einen Maximalwert von 25 Hertz vor, sodass die Nachrichtenübertragung in Zyklen von 40 Millisekunden erfolgen kann.
Da eine weitere spezifizierte Anforderung die höchstmögliche Auslastung des Bussystems vorsieht, ist es erforderlich die minimale Größe eines Zeitschlitzes zu berechnen, die notwendig ist, um eine Nachricht in einem FlexRay-System zu übertragen. Aus der Anzahl der zu übertragenden Nachrichten, für die jeweils ein Zeitschlitz benötigt wird, kann die Zykluslänge anschließend abgeleitet werden.
Für die Größenbestimmung eines Zeitschlitzes ist die Signallaufzeit einer Nachricht auf dem Kommunikationsmedium ausschlaggebend, wobei die größtmögliche Entfernung zwischen zwei kommunizierenden Netzwerkknoten anzusetzen ist.
Formel (4.2) zeigt die Berechnung eines Zeitschlitzes des TDMA-Ablaufplans, wobei die Größe der Payload mit 8 Byte veranschlagt wird, um die möglichen Wertebereiche der Sensoren abzudecken. Die Berechnung ist auf die Konfiguration eines FlexRay-Systems bezogen, wobei ein Zeitschlitz der statischen Slotgröße entspricht.

Die verwendeten Einheiten Byte und Mikrosekunden können in der Berechnung für die Nachrichtengröße gleichbedeutend eingesetzt werden. Bei FlexRay wird durch die Protocol Engine der Kommunikationscontroller eine NRZ-Codierung angewendet, um die Nachrichten für die Übertragung vorzubereiten. Jedem zu übertragenden Byte wird die sogenannte Byte Start Sequence vorangestellt, die aus einem High-Bit und einem Low-Bit besteht und den Empfängern zur Synchronisierung auf den Beginn der übertragenden Bytes dient. Somit wird ein

Byte für den Sendevorgang mit 10 Bit codiert, was bei einer Datenrate von 10 MBit/s genau einer Mikrosekunde entspricht [Raus2008].

$$\begin{aligned} Nachrichtengröße &= Payload + Header + CRC \\ &= 8Byte + 5Byte + 3Byte = \underline{\underline{16Byte}} \end{aligned} \tag{4.1}$$

$$\begin{aligned} Statische\ Slotgröße &= Nachrichtengröße + IdleErkennungszeit + Sicherheitszuschlag \\ &= 16\mu s + 1{,}1\mu s + 13{,}9\mu s = \underline{\underline{31\mu s}} \end{aligned} \tag{4.2}$$

Die in der Berechnung aufgeführte Idle-Erkennungszeit wird von den Kommunikationsteilnehmern benötigt, um nach dem Senden eines Frames zu erkennen, dass keine Aktivität mehr auf den Kommunikationskanälen ist und dass der nächste Static-Slot beginnt. Der Wert von 1,1 Mikrosekunden ergibt sich aus einer im Protokoll fest vorgegebenen Konstante.
Der Sicherheitszuschlag, der in der Berechnung aufgeführt ist und dessen Größe mit Hilfe eines Werkzeugs dimensioniert wurde, fasst die folgenden Faktoren zusammen [Raus2008]:

- Ein zeitlicher Zuschlag für die Aktivierung des Sende- beziehungsweise Empfangsvorgangs
- Ein Sicherheitszuschlag, der die Uhrenabweichungen zwischen Sender und Empfängern berücksichtigt
- Ein Zuschlag für die Signallaufzeit, die je nach Entfernung zwischen den Kommunikationsteilnehmern variiert

Das Ergebnis der dargestellten Berechnung zeigt ausschließlich die Zeit, die für die Nachrichtenübertragung benötigt wird. Bevor der Kommunikationscontroller eines FlexRay-Knoten gültige Daten übertragen kann, müssen ihm diese von der auf dem Host ausgeführten Applikation zur Verfügung gestellt werden. Hierfür werden die Daten über das Controller-Host-Interface in einem Sendepuffer des Kommunikationscontrollers gespeichert und zum Zeitpunkt des Sendens von der Protocol Engine geholt (vgl. Kapitel 3.2). Für den Entwurf und die Kalkulation der Zeitschlitze beziehungsweise der statischen Slotgröße muss daher die Zeit, die der Host zum Bereitstellen der Daten benötigt, berücksichtigt werden.
Des Weiteren wird mit dem resultierenden Kommunikationszyklus das Raster für die Zykluszeit der Applikation vorgegeben, sodass die Ausführungszeiten weiterer Aktivitäten, die auf dem Host ausgeführt werden sollen, einzubeziehen sind.
Das heißt, dass der Entwurf eines TDMA-Ablaufplans die Sendevorgänge, sowie das zugehörige Breitstellen der Daten ermöglichen kann, aber keine Zeit für die Ausführung anderer Ak-

tivitäten bleiben würde. Somit ist es erforderlich, dass das Ausführungsintervall und die Ausführungsdauer der Aktivitäten vorab bekannt sind oder abgeschätzt werden.
Für die Bestimmung der maximalen Ausführungszeiten werden dynamische und statische Analysen unterschieden.
Bei der statischen Analyse wird die maximale Ausführungszeit des Programmcodes analytisch bestimmt, ohne ihn auf der Zielhardware auszuführen. Dieser Ansatz liefert exakte Ergebnisse, da die Ausführungszeit jeder Instruktion eines Ausführungspfades berücksichtigt wird [Ring2002]. In der vorliegenden Arbeit sind die Ausführungszeiten der Aktivitäten durch eine dynamische Analyse in Form von Messungen bestimmt worden, da der Quellcode einiger verwendeter Komponenten nicht zugänglich und somit die Voraussetzung für eine statische Analyse nicht gegeben war.
Bei beiden Ansätzen ist es erforderlich den linken Ast des V-Modells (vgl. Abbildung 19) zunächst mit einem Grobentwurf bis zur Komponenten-Implementierung zu durchlaufen. Mit dem aus den Analysen resultierenden Feinentwurf kann die Cluster-Spezifikation modifiziert und die Einhaltung der Anforderungen sichergestellt werden.

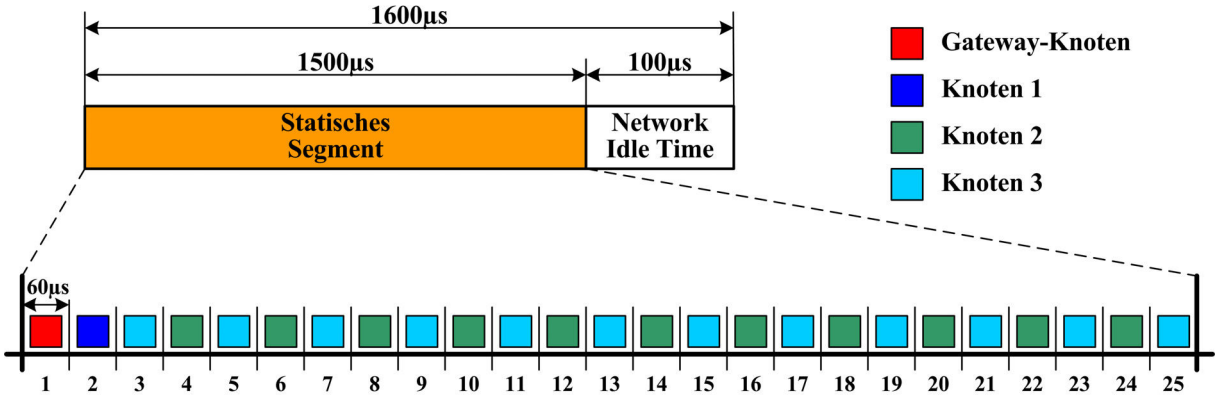

Abbildung 21 – Entwurf des FlexRay-Kommunikationszyklus

In Abbildung 21 ist der Entwurf des FlexRay-Kommunikationszyklus dargestellt, der aus den folgenden Entwurfsentscheidungen und Einflusskriterien resultiert:

- Der Kommunikationszyklus setzt sich ausschließlich aus dem Statischen Segment und der Network Idle Time zusammen. Das Dynamische Segment und das Symbol Window werden für die spezifizierten Anforderungen nicht benötigt.
- Die Größe der Static-Slots wurde von den berechneten 31 auf 60 Mikrosekunden verlängert. Die zusätzliche veranschlagte Zeit ermöglicht eine flexiblere Gestaltung der Taskaktivierungszeitpunkte und bietet die Möglichkeit zur Realisierung von rechenintensiveren Aktivitäten.

- Die Systemanforderungen erfordern die Übertragung von 25 Nachrichten, sodass die Konfiguration der gleichen Anzahl Static-Slots notwendig ist. Die Zuordnung der Static-Slots zu den Knoten ist durch farbige Markierungen gekennzeichnet. Für das Statische Segment ergibt sich daraus eine Gesamtlänge von 1500 Mikrosekunden.
- Die berechnete Wiederholungsrate der Nachrichtenübertragung von 40 Millisekunden ist auf 1600 Mikrosekunden verkürzt, um für den Testaufbau eine höchstmögliche Last erzeugen zu können. Durch die messtechnische Analyse ist sichergestellt, dass ausreichend Zeit für zusätzliche Aktivitäten auf den Knoten bleibt.

Mit dem Entwurf des TDMA-Ablaufplans beziehungsweise FlexRay-Kommunikationszyklus sind alle notwendigen Arbeitsschritte des Cluster-Designs abgeschlossen. Für den Übergang zur Komponenten-Ebene muss für jeden Knoten eine Konfiguration erstellt werden, die jeweils nur die relevanten Sende- und Empfangszeitpunkte berücksichtigt. In dieser Arbeit wird dafür das Werkzeug DECOMSYS::DESIGNER verwendet, dessen Verwendung im Anhang in Kapitel A.3.1 genau erläutert wird und mit dem die Konfigurationen automatisch erstellt und generiert werden können.
In Abbildung 22 ist die Cluster-Konfiguration im DECOMSYS::DESIGNER mit den zuvor berechneten Werten dargestellt.

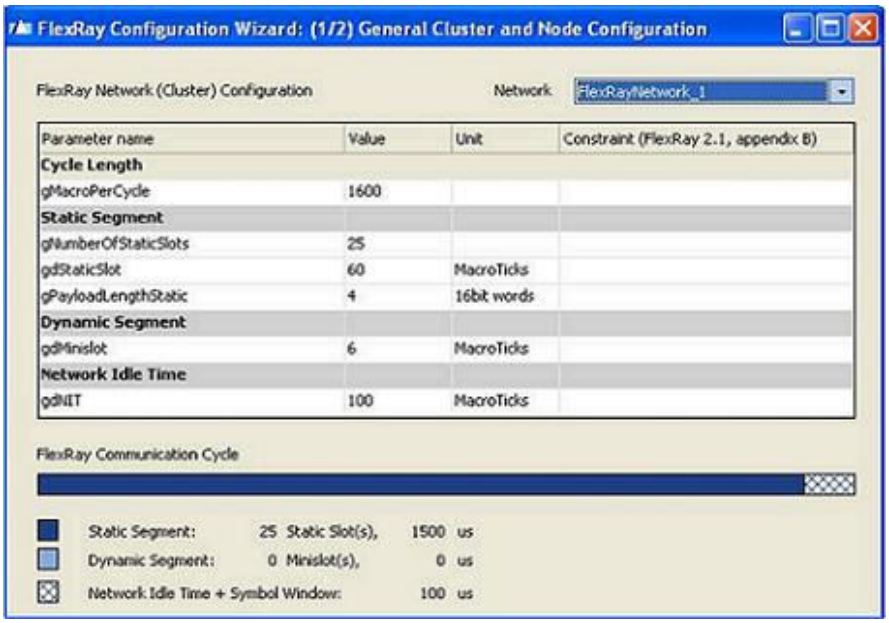

Abbildung 22 – Cluster-Spezifikation im DECOMSYS::DESIGNER

Mit solchen Konfigurationswerkzeugen können die Netzwerkdesigndaten im sogenannten FIBEX-Format (Field Bus Exchange Format) exportiert werden und ermöglichen so den Austausch der Daten und damit den beschriebenen parallelen Entwicklungsprozess beim Komponentendesign.

5. Design und Realisierung der Komponenten

Durch die im vorangegangenen Kapitel fertig gestellte Cluster-Spezifikation kann der nächste Arbeitsschritt des zweistufigen Entwicklungsansatzes folgen, der das Design und die Realisierung der Komponenten beinhaltet (vgl. Abbildung 19).
Die notwendige Koordination der Kommunikationsteilnehmer ist durch die Spezifikation der Sendezeitpunkte ebenfalls gegeben und wird in der Entwicklungsphase des Komponentendesigns durch den fortführenden Einsatz des DECOMSYS::DESIGNER Werkzeugs sichergestellt. Dadurch kann für jeden Knoten garantiert werden, dass der Nachrichtenaustausch gemäß der Cluster-Spezifikation durchgeführt wird und damit eine Überlastung des Kommunikationssystems ausgeschlossen ist [Tech2007].
In diesem Kapitel werden die Aufgaben und die Implementierung der einzelnen Knoten näher beschrieben und zwei verschiedene Betriebssysteme vorgestellt, die für die Ausführung der Aktivitäten auf den Netzwerkknoten eingesetzt werden. Anschließend werden die beiden Lösungswege verglichen und bewertet.

5.1 Verwendete Hardware

In der vorliegenden Bachelorarbeit werden für den Aufbau des Kommunikationssystems die in Abbildung 23 dargestellten Netzwerkknoten der Firma Decomsys verwendet.

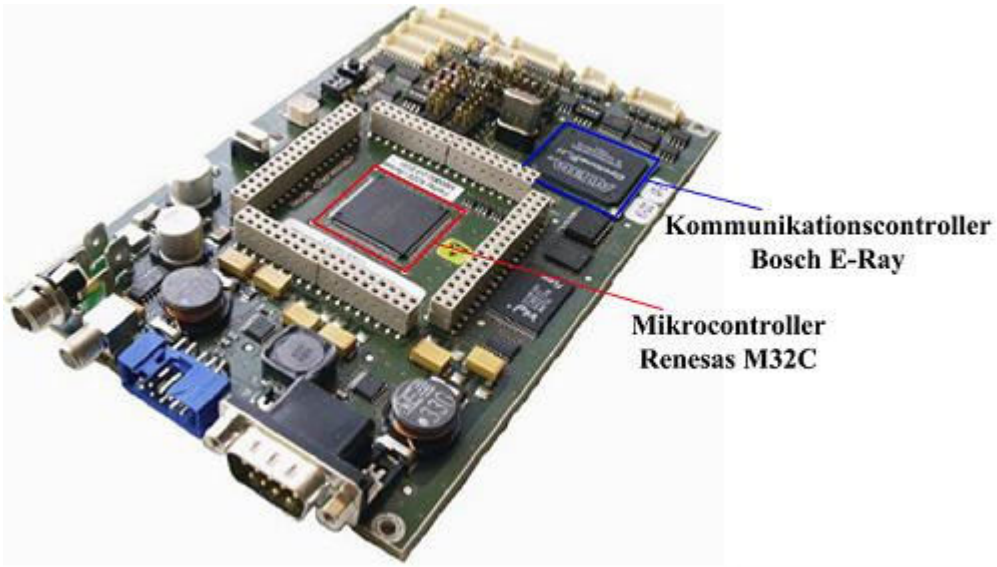

Abbildung 23 - Evaluierungs-Plattform für FlexRay-Anwendungen

Ein Knoten setzt sich aus einem 24 MHz-Mikrocontroller vom Typ M32C der Firma Renesas und einem Bosch E-Ray Kommunikationscontroller zusammen, für die jeweils ein eigener Oszillator zur Taktgenerierung vorgesehen ist. Zur Ausstattung der Plattform gehören 2 MB Flash, zwei FlexRay Kanäle, zwei CAN Kanäle, eine LIN Schnittstelle, eine RS232 Schnittstelle sowie eine Reihe digitaler und analoger Ein- und Ausgänge [De1_2006].
Zum Download des Programmcodes auf die Knoten wird der Renesas KD3083-Debugger eingesetzt, mit dem die Möglichkeit gegeben ist, die jeweilige Anwendung über eine JTAG-Schnittstelle auf dem Mikrocontroller zu debuggen.
Der Kommunikationscontroller wird nach dem Einschalten eines Knotens über eine Schnittstelle durch die auf dem Mikrocontroller ausgeführte Anwendung initialisiert und konfiguriert. Während der Konfiguration wird der TDMA-Ablaufplan bereitgestellt, den der Kommunikationscontroller anschließend vollkommen unabhängig abarbeitet. Das bedeutet, dass er zu den festgelegten Sendezeitpunkten die Nutzdaten, die ebenfalls von der Anwendung bereitgestellt werden müssen, aus den dafür vorgesehenen Puffern holt und die Nachrichtenübertragung selbstständig durchführt.

5.2 Software-Design

Die für die Netzwerkknoten zu entwickelnde Software setzt sich aus den folgenden drei Modulen zusammen:

- Commstack-Konfiguration
- Betriebssystem-Konfiguration
- Anwendungssoftware

Der Commstack ist eine Software-Bibliothek und stellt eine Abstraktionsschicht für den Zugriff auf die Kommunikationscontroller dar. Durch die zur Verfügung gestellten Funktionen wird unter anderem der Datenaustausch zwischen der Anwendung und dem Kommunikationscontroller über die Sende- und Empfangspuffer ermöglicht. Mit der zu erstellenden Commstack-Konfiguration kann der Kommunikationscontroller durch die Anwendung konfiguriert und der TDMA-Ablaufplan bereitgestellt werden.
Die Betriebssystem-Konfiguration beinhaltet hauptsächlich die Erstellung der sogenannten Dispatcher-Tabelle, in der die Tasks und die zugehörigen Aktivierungszeitpunkte der jeweiligen Knoten zusammengefasst sind.
Sowohl die Betriebssystem-Konfiguration, die im Kapitel 5.4 genau erläutert wird, als auch die Commstack-Konfiguration können mit Hilfe des DECOMSYS::DESIGNER definiert und anschließend der daraus resultierende C-Code mit dem Werkzeug generiert werden.

Die Anwendungssoftware umfasst die Implementierung der Tasks, mit denen das gewünschte Verhalten der Netzwerkknoten realisiert wird. Für die Implementierung wird in der vorliegenden Arbeit eine Entwicklungsumgebung der Firma IAR Systems verwendet. Anschließend werden die beiden generierten Module integriert und zusammen mit dem Applikationscode kompiliert. Mit dem Linker der Entwicklungsumgebung folgt die Verbindung der einzelnen Module zum ausführbaren Programmcode.

5.3 Applikationszyklus und Kommunikationszyklus

Mit der Definition der Zykluszeit des Kommunikationszyklus ist auch ein zeitliches Raster für die Anwendung vorgegeben, da die Planung der Aktivierungszeitpunkte der Tasks an der globalen Uhrzeit des Kommunikationssystems ausgerichtet wird. Das bedeutet, dass ein Task, der beispielsweise Daten für einen Sendevorgang bereitstellt, abgeschlossen sein muss, bevor der dafür vorgesehene Sendezeitpunkt des Kommunikationszyklus erreicht ist.
Die zyklische Abarbeitung der Dispatcher-Tabelle durch die Anwendung und die damit verbundene Aktivierung der Tasks wird daher als Applikationszyklus bezeichnet. Dabei muss die Länge des Applikationszyklus nicht zwingend der Zykluszeit des Kommunikationssystems entsprechen, sondern kann sich auch aus Vielfachen dieser Zeit zusammensetzen.
Des Weiteren müssen die Zeiten des Kommunikationszyklus und des Applikationszyklus synchronisiert werden, da ansonsten das Auseinanderlaufen der unabhängigen Uhren dazu führen würde, dass Sende- oder Empfangszeitpunkte nicht mehr eingehalten werden können. Die notwenige Synchronisierung wird in dieser Arbeit durch einen Task realisiert, der bei der Planung der Dispatcher-Tabelle berücksichtigt werden muss. Die Aktivierung dieses Tasks erfolgt typischerweise zur Network Idle Time des Kommunikationszyklus, da in diesem Abschnitt des Zyklus keine Kommunikation stattfindet und die Synchronisierung der Kommunikationscontroller durchgeführt wird.

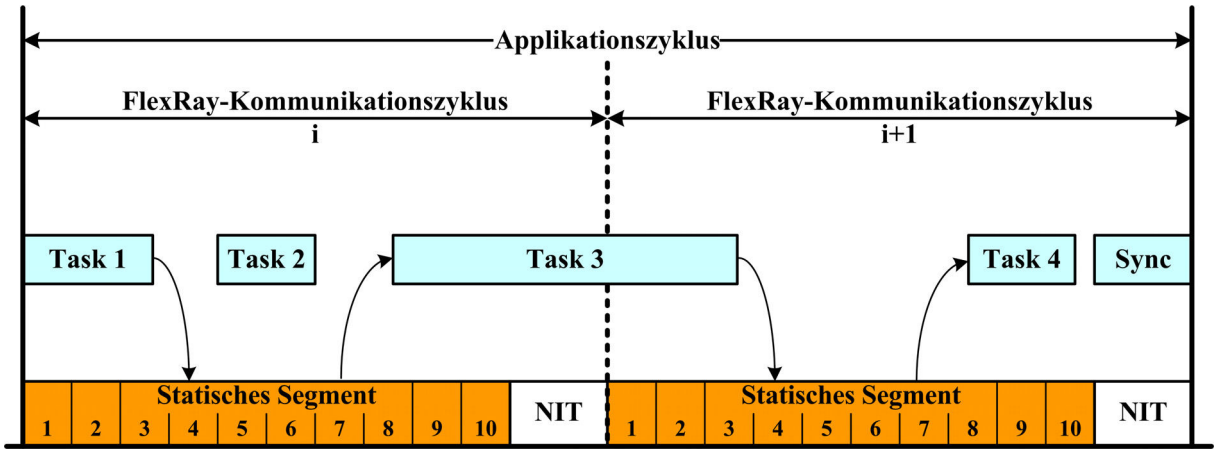

Abbildung 24 – Exemplarischer Entwurf eines Applikationszyklus

Für die Ausführung des Synchronisierungs-Tasks wird Rechenzeit in Anspruch genommen, die für die Realisierung der eigentlichen Aufgaben nicht mehr zur Verfügung steht. Daher kann es für die Umsetzung der jeweiligen anwendungsspezifischen Anforderungen sinnvoll sein, die Zykluszeit des Applikationszyklus im Verhältnis zum Kommunikationszyklus zu vervielfachen.
Abbildung 24 stellt einen exemplarischen Entwurf einer Anwendung dar, die für eine rechenintensive Datenbearbeitung eingesetzt wird. Der Zeitraum zwischen dem Empfang dieser Daten und dem Weiterleiten des Berechnungsergebnisses wird vollständig von dem rechenintensiven Task benötigt, sodass der Synchronisierungs-Task erst zum Ende des zweiten Kommunikationszyklus ausgeführt werden kann.
Die Grundvoraussetzung für eine solche Vervielfachung der Zykluszeit ist, dass trotz des vergrößerten Ausführungsintervalls des Synchronisierungs-Tasks eine ausreichende Synchronität zum Zeitverständnis des Kommunikationssystems gewährleistet bleibt.

5.4 Betriebssysteme

Ein Betriebssystem übernimmt in einem eingebetteten, verteilten Echtzeitsystem die Verwaltung der zuvor definierten Tasks und deren Ausführungszeitpunkte. Zur Laufzeit erfolgt ein statisches Aktivieren der Tasks zu den geplanten Zeitpunkten durch einen sogenannten Dispatcher, der einen Teil des Betriebssystems darstellt [Ring2002].
Des Weiteren muss eine Möglichkeit gegeben sein, die lokale Uhrzeit nach der das Betriebssystem arbeitet, an die globale Uhrzeit des FlexRay-Systems anpassen zu können, da die vorab geplante Kommunikation und damit verbundene Koordination der Netzwerkteilnehmer auf dem gemeinsamen Zeitverständnis beruht.
In den folgenden Abschnitten werden zwei Betriebssysteme vorgestellt und anschließend deren Vor- und Nachteile gegenübergestellt.

5.4.1 OSEKtime

OSEKtime ist ein kommerzielles, zeitgesteuertes Echtzeitbetriebssystem für eingebettete Systeme, dass von zahlreichen Automobil-Herstellern, auch im Zusammenhang mit FlexRay verwendet wird [Sche2007]. Es handelt sich um ein Multitasking-Betriebssystem, das die Abarbeitung mehrerer Funktionsstränge erlaubt und dafür drei verschiedene Zustände für Tasks vorsieht. In Abbildung 25 sind die Zustände und Zustandsübergänge dargestellt.

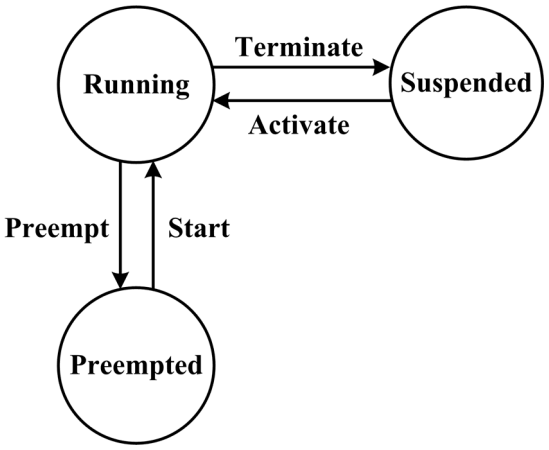

Abbildung 25 – Zeitgesteuerte Tasks in OSEKtime [OSEK2001]

Im Zustand ***running*** wird dem entsprechenden Task Prozessorzeit zur Abarbeitung seiner Instruktionen zugeteilt. Diesem Zustand kann zu jedem Zeitpunkt jeweils nur ein Task zugeteilt sein, während die anderen beiden Zustände von mehreren Tasks gleichzeitig eingenommen werden können.

In den Zustand ***preempted*** geht ein Task über, dessen Ausführung noch nicht abgeschlossen ist und unterbrochen werden muss, da der Aktivierungszeitpunkt eines anderen Tasks erreicht ist. Sobald die Instruktionen dieses Tasks abgearbeitet sind, veranlasst der Dispatcher die Rückkehr des unterbrochenen Tasks in den Zustand *running*.

Im Zustand ***suspended*** ist der Task passiv und kann zum nächsten geplanten Zeitpunkt aktiviert werden.

Die Abarbeitung der Tasks erfolgt in OSEKtime durch einen Dispatcher, der durch die Interrupts eines Timer aktiviert wird. Dazu werden vor der Laufzeit des Systems die Aktivierungszeitpunkte der Tasks festgelegt und in einer sogenannten Dispatcher-Tabelle abgelegt. Die Tabelle wird zyklisch durchlaufen und der Timer des Mikrocontrollers jeweils auf den Wert der nächsten Aktivierung eingestellt, wobei jeder Durchlauf der Tabelle als Dispatcher-Runde bezeichnet wird.
Die Planung der Aktivierungszeitpunkte ist ein entscheidendes Kriterium für die Leistungsfähigkeit des zu entwickelnden Systems, da bei zu kurz gewählten Abständen mehrere Tasks zur Ausführung anstehen würden und dies möglicherweise zu Verzögerungen des Gesamtsystems führen würde. Damit die Ausführbarkeit der Tasks gewährleistet werden kann, ist eine genaue Kenntnis der maximal möglichen Laufzeit jeder einzelnen Task erforderlich. Neben Laufzeitmessungen im System ist dafür eine statische Analyse des Programm-Codes notwendig, um sichere Werte zu erhalten [Homa2004] [Scho2001].

Durch das preemptive Konzept von OSEKtime kann eine bessere Ausnutzung der Rechenzeit durch eine entsprechende Planung bei dem Entwurf der Dispatcher-Tabelle erreicht werden. Die Planung kann beispielsweise so ausgelegt sein, dass ein Task, dessen Ausführung viel Rechenzeit in Anspruch nimmt, durch einen Task mit einer kurzen Ausführungszeit unterbrochen wird.
In Abbildung 26 ist ein solches Szenario exemplarisch dargestellt. Es ist ersichtlich, dass eine Aufteilung des rechenintensiven Tasks A in mehrere Teilaufgaben dazu führt, dass Rechenzeit ungenutzt bleibt. Dagegen kann eine bessere Ausnutzung der Rechenzeit durch eine kurzzeitige Unterbrechung der Task erreicht werden [Ring2002].

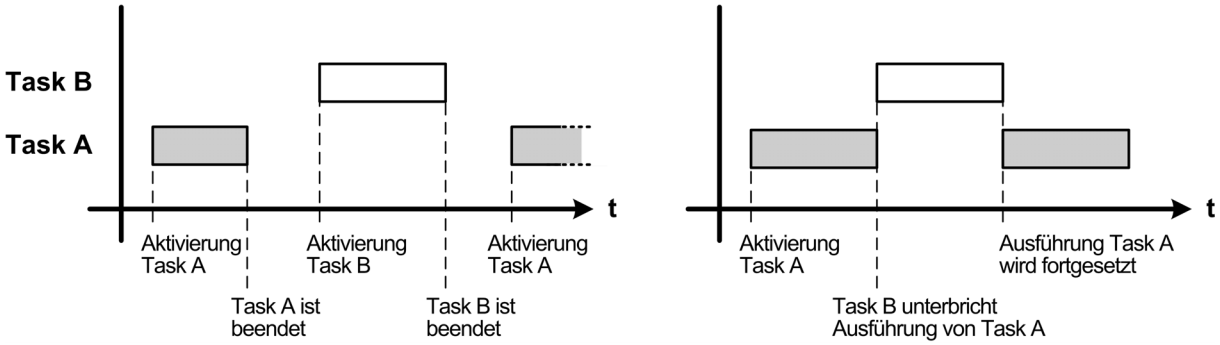

Abbildung 26 – Verbesserte Nutzung der Rechenzeit [Ring2002]

Eine weitere Möglichkeit zur verbesserten Ausnutzung der Rechenzeit ist durch die Verwendung des sogenannten Idle-Tasks gegeben. Der Idle-Task wird immer dann zur Ausführung gebracht, wenn ein zeitgesteuerter Task bereits beendet ist und der Aktivierungszeitpunkt des darauffolgenden Tasks noch nicht erreicht ist. Diese freie Zeit kann durch die Verwendung der Idle-Task ausgefüllt und so Aktivitäten umgesetzt werden, für die keine zeitlichen Anforderungen vorgesehen sind.
Des Weiteren wird durch das Betriebsystem die Bearbeitung von Aufgaben unterstützt, die zu vorab nicht bekannten Zeitpunkten durchzuführen sind und somit keine expliziten Aktivierungszeitpunkte angegeben werden können. Für die Reaktion auf solche dynamischen Ereignisse können Interrupt Service Routinen (ISR) verwendet werden. Da der Aufruf von Interrupt Service Routinen asynchron zur laufenden Anwendung erfolgt, ist bei der Planung, neben der maximalen Ausführungszeit, die minimale Zeit zu berücksichtigen, die zwischen zwei Aufrufen einer ISR liegen kann. Nur wenn beide Zeiten mit Sicherheit bestimmt werden können, ist gewährleistet, dass alle Aufgaben in dem geplanten Zyklus abgearbeitet werden können und das System nicht überlastet wird.

Deadline-Überwachung
Das OSEKtime Betriebssystem stellt einen Fehlererkennungs-Mechanismus zur Verfügung, mit dem die Ausführung der definierten Tasks überwacht werden kann. Während der Konfiguration des OSEKtime kann zu jedem Task eine sogenannte Deadline spezifiziert werden, die den Zeitpunkt beschreibt, an dem die Ausführung unter Berücksichtigung aller möglichen Unterbrechungen durch andere Tasks oder Interrupts beendet sein muss. Die festgelegten Deadlines werden in der Dispatcher-Tabelle gespeichert und die Einhaltung durch den Dispatcher überwacht. Die Verletzung einer Deadline hat zur Folge, dass das erforderliche deterministische Verhalten des Echtzeitsystems nicht weiter garantiert werden kann und eine Fehlerbehandlung einzuleiten ist. Für diesen Fall ist eine spezielle Funktion vorgesehen, die durch das Betriebssystem aufgerufen wird und durch die anwendungsspezifische Aktivitäten durchgeführt werden können. Nach dem Verlassen dieser Funktion wird das System heruntergefahren.

Synchronisation mit dem FlexRay-Netzwerk
Die Synchronisation mit den Kommunikationsteilnehmern des FlexRay-Netzwerks wird in dieser Arbeit durch die Verwendung von Software-Bibliotheken realisiert. Dafür muss bei der Konfiguration von OSEKtime und der damit verbundenen Erstellung der Dispatcher-Tabelle ein zusätzlicher Task eingeplant werden. Dieser Task hat die Aufgabe, die global synchronisierte Uhrzeit des FlexRay-Netzwerks in jeder Dispatcher-Runde bei dem knotenlokalen Kommunikationscontroller zu erfragen. Mit Hilfe dieser Uhrzeit kann darauf die Anpassung der lokalen Uhrzeit durch eine Verkürzung beziehungsweise Verlängerung der Dispatcher-Runde erfolgen. Bei der Verwendung der Software-Bibliotheken sind zwei Funktionsaufrufe notwendig, mit denen zunächst überprüft wird, ob der jeweilige Kommunikationscontroller zum FlexRay-Netzwerk synchronisiert ist und somit eine korrekte Uhrzeit zur Verfügung stellen kann. Anschließend folgt die Synchronisierung des Mikrocontrollers durch den Aufruf einer weiteren API-Funktion der Software-Bibliothek.

Konfiguration von OSEKtime
Die Konfiguration von OSEKtime erfolgt über eine grafische Benutzeroberfläche, die sowohl die Erstellung neuer Projekte erlaubt, als auch den Import von Dateien, die mit der speziellen Beschreibungssprache OSEK Implementation Language (OIL) erstellt worden sind.
Von dem für die FlexRay-Konfiguration verwendeten Werkzeug DECOMSYS::DESIGNER wird diese Beschreibungssprache als Ausgabeformat unterstützt, sodass auch die Spezifikation der Tasks und deren Aktivierungszeitpunkte mit dem Werkzeug durchgeführt werden kann. Anschließend kann die vom DECOMSYS::DESIGNER erstellte OIL-Datei mit Hilfe des ProOSEK-Configurators importiert und im nächsten Schritt der Quellcode des Betriebssystems generiert werden (vgl. Abbildung 27).

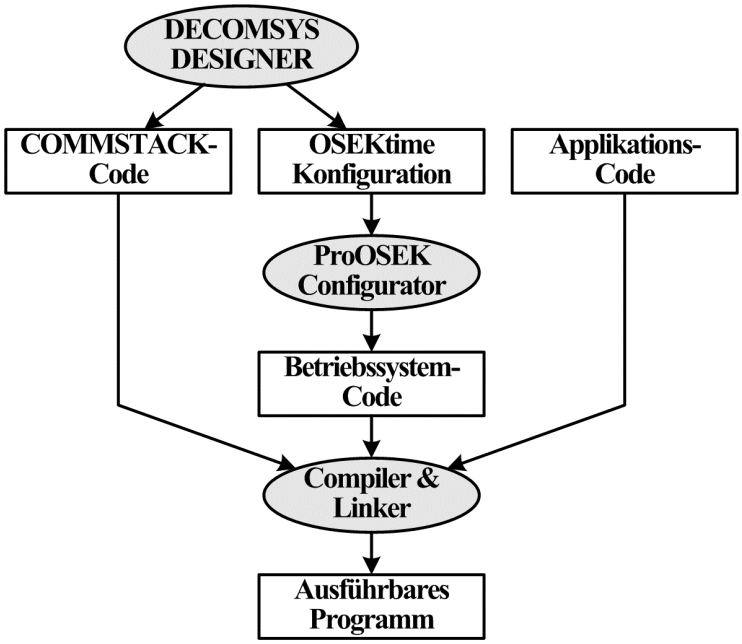

Abbildung 27 – Software-Design mit OSEKtime

Des Weiteren besteht die Möglichkeit alle genannten Informationen zu Tasks und ISRs, die in die Konfiguration eingegangen sind, mit dem ProOSEK-Configurator auf Vereinbarkeit zu überprüfen und im Fehlerfall entsprechende Änderungen vorzunehmen.

5.4.2 Decomsys - Application Execution System

Das Application Execution System (AES) der Firma Decomsys stellt ebenfalls ein zeitgesteuertes Echtzeitbetriebssystem dar, das auf der Grundlage eines Hardware-Timers die Aktivierung des Dispatchers realisiert.
In Abbildung 28 sind die erforderlichen Schritte von der Konfiguration mit dem DECOSYS:: DESIGNER bis zum ausführbaren Programm dargestellt.

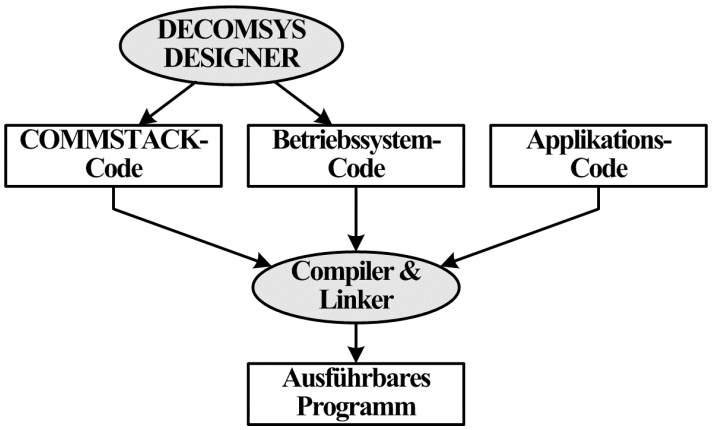

Abbildung 28 – Software-Design mit AES

Im Gegensatz zu OSEKtime können im Zuge der Konfiguration ausschließlich die anwendungsspezifischen Tasks und die zugehörigen Aktivierungszeitpunkte definiert werden. Interrupt Service Routinen und die Überwachung der Deadlines werden durch das Betriebssystem nicht unterstützt. Des Weiteren muss durch den Anwender sichergestellt werden, dass die geplanten Ausführungszeiten der Tasks durch das Betriebssystem umgesetzt werden können und die daraus resultierende Dispatcher-Tabelle die folgenden Randbedingungen [De2_2006] einhält:

- Die Einträge müssen im Hinblick auf die Aktivierungszeit fortlaufend geordnet sein.
- Es ist nicht zulässig mehrere Tasks mit der gleichen Aktivierungszeit zu definieren. Somit ist auch eine Konfiguration unzulässig, die einen Task mit der Ausführungszeit 0 vorsieht und einen anderen mit einer Ausführungszeit, die der Zykluslänge entspricht.
- Für jeden Eintrag muss die entsprechende Implementierung des Tasks vorhanden sein.
- Die Ausführungszeit des letzten Task-Eintrags muss kleiner als die Zykluslänge sein und nach der Beendigung der Ausführung muss ausreichend Zeit verfügbar sein, um die korrekte Synchronisation mit dem Netzwerk innerhalb des aktuellen Zyklus zu gewährleisten.

Das bedeutet, dass mit Hilfe des Werkzeugs keine Verifikation der Dispatcher-Tabelle durchgeführt werden kann, sodass es erforderlich ist, die Einhaltung der Randbedingungen, die für den Einsatz des Betriebssystems vorgegeben sind, im generierten C-Code zu überprüfen und gegebenenfalls Einträge per Hand einzufügen oder zu bearbeiten.
Ein weiterer Unterschied zum beschriebenen OSEKtime Betriebssystem stellen die möglichen Task-Zustände dar. Im AES sind die zur Ausführung gebrachten Tasks nicht unterbrechbar und somit erfolgt die Abarbeitung ausschließlich in der Reihenfolge ihrer Aktivierung.
In diesem Kontext erfolgt auch die Aktivierung des Idle-Tasks, der entsprechend zu OSEKtime implementiert werden kann, um die Rechenzeit zu nutzen, in der keine zeitgesteuerten Tasks zur Ausführung anstehen.

Synchronisation mit dem FlexRay-Netzwerk
Die Synchronisation des AES erfolgt wie bereits in Kapitel 5.4.1 beschrieben, durch die Verwendung von Software-Bibliotheken, die die synchronisierte Zeitbasis des FlexRay-Busses nutzen. Über das Controller-Host-Interface des jeweiligen Kommunikationscontrollers kann die aktuelle Uhrzeit des Netzwerks abgefragt und die lokale Uhrzeit des Betriebssystems angepasst werden.

5.4.3 Bewertung der Betriebssysteme

In der vorliegenden Arbeit ist das Design und die Implementierung der Komponenten sowohl mit OSEKtime als auch mit dem Application Execution System umgesetzt worden.
In beiden Realisierungsvarianten ist die Synchronisierung der lokalen Uhren der Mikrocontroller mit dem Einsatz von Software-Bibliotheken implementiert worden. Die darauffolgende messtechnische Analyse ergab in Hinblick auf das Gesamtsystem, dass das im Entwurf geplante Verhalten umgesetzt werden konnte und damit in beiden Fällen keine Einschränkungen der Funktion zu beobachten ist. Die Synchronität der verteilten Netzwerkknoten und das gleiche Zeitverständnis für den Start und das Ende jedes Applikationszyklus ist in Abbildung 29 für das AES und in Abbildung 30 für das OSEKtime Betriebssystem dargestellt.
Eine statische Analyse des Quellcodes, insbesondere der generierten Dispatcher-Tabellen zeigt hingegen, dass die Aktivierungszeitpunkte der Tasks unter OSEKtime jeweils relativ zum Startzeitpunkt des vorigen Tasks gespeichert werden, sodass dem Timer bei einem Taskwechsel direkt der nächste Aktivierungszeitpunkt aus der Dispatcher-Tabelle zugewiesen werden kann.
In der Dispatcher-Tabelle des AES sind die Aktivierungszeitpunkte relativ zum Startzeitpunkt der Dispatcher-Runde gespeichert, sodass eine Berechnung erforderlich ist, mit der der nächste Timer-Wert bestimmt wird. Dafür muss zunächst die aktuelle Zeit der Dispatcher-Runde ermittelt werden und vom Aktivierungszeitpunkt des Tasks subtrahiert werden. So wird bei

einem Taskwechsel zusätzliche Rechenzeit vom Dispatcher beansprucht und die Aktivierung des Tasks verzögert.

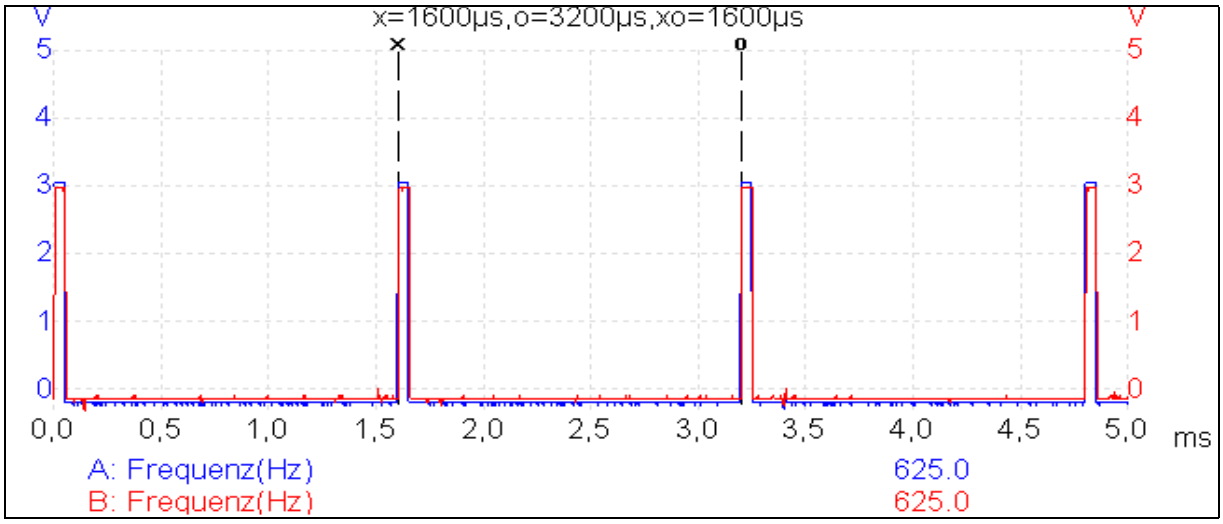

Abbildung 29 – Synchronität zweier Knoten unter Verwendung des AES

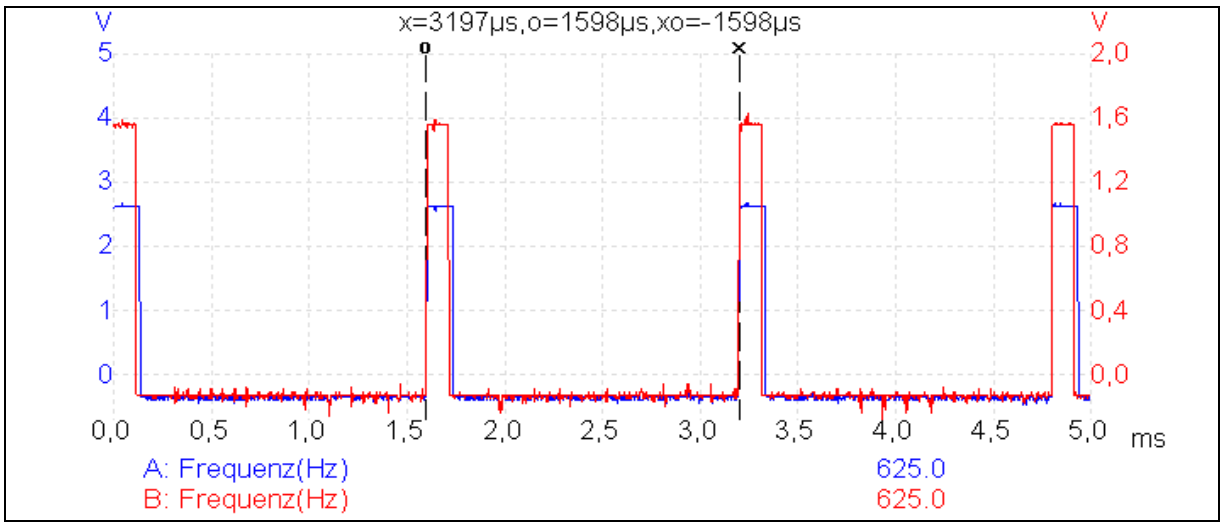

Abbildung 30 – Synchronität zweier Knoten unter Verwendung von OSEKtime

Weitere Vorteile des OSEKtime Betriebssystem sind die beschriebenen Fehlererkennungsmechanismen durch die Angabe von Deadlines, sowie die Verifikation der festegelegten Aktivierungszeitpunkte und der maximalen Ausführungszeiten.
Die letztendliche Entscheidung für einen Lösungsweg ist zum einen von der Komplexität des zu realisierenden Systems und zum anderen von den anwendungsspezifischen Anforderungen abhängig. Bei einem System, das für sicherheitsrelevante Steuerungsaufgaben eingesetzt wer-

den soll und eine Deadline-Überwachung zwingend erfordert, ist der benötigte Arbeitsaufwand für eine individuelle Implementierung ein gewichtiger Kostenfaktor, sodass eine Kostenanalyse anzusetzen ist, die die Lizenzkosten einer kommerziellen Lösung wie OSEKtime gegenüberstellt und auswertet.

5.5 Realisierung der Knoten

Für die Realisierung der Knoten wird die in Kapitel 4.3 geplante Aufteilung umgesetzt. Für den Versuchsaufbau im Labor werden die den Knoten zugeteilten Sensoren durch Software-Module simuliert, um so die realen Verhältnisse nachzubilden. In den folgenden Abschnitten wird der Entwurf der Software-Module erläutert und auf die zu berücksichtigenden Randbedingungen eingegangen. Abschließend wird die Task-Aufteilung der Netzwerkknoten innerhalb des Applikationszyklus beschrieben und die benötigten Aktivierungszeitpunkte definiert. Die damit festgelegte Komponentenspezifikation stellt den letzten Planungsschritt des zweistufigen Entwicklungsansatzes dar und bildet die Grundlage für die darauffolgende Implementierung der Komponenten und den Abschluss der Entwicklung mit der Integration ins Gesamtsystem.

5.5.1 Sensorwerte der Motorsteuerung

Die bei der Kommunikationsplanung vorgenommene Zuteilung der Nachrichten zu bestimmten Knoten erfordert, dass für zwei Netzwerkknoten Software-Module implementiert werden, die die Sensorwerte der Motorsteuerung für den Versuchsaufbau simulieren. Für die Simulation müssen die möglichen Wertebereiche der jeweiligen Sensoren [Pasc2007] berücksichtigt werden und eine Möglichkeit zur Konfiguration der zugehörigen Aktualisierungsraten gegeben sein. Die generierten Werte werden der Kommunikationsplanung entsprechend über den FlexRay-Bus an den Gateway-Knoten übertragen.

5.5.2 Raddrehzahl

In Abbildung 31 ist die Funktionsweise eines im Formula Student Rennwagen eingesetzten Drehzahl-Sensors dargestellt, der während einer Radumdrehung 48 Rechteckimpulse generiert. Die Modellierung des Drehzahl-Sensors ist für den Versuchsaufbau durch einen Hardware-Timer realisiert, mit dem ein Rechtecksignal erzeugt wird und so die möglichen Frequenzbereiche, die bei unterschiedlichen Radumlaufgeschwindigkeiten auftreten können, umsetzt.

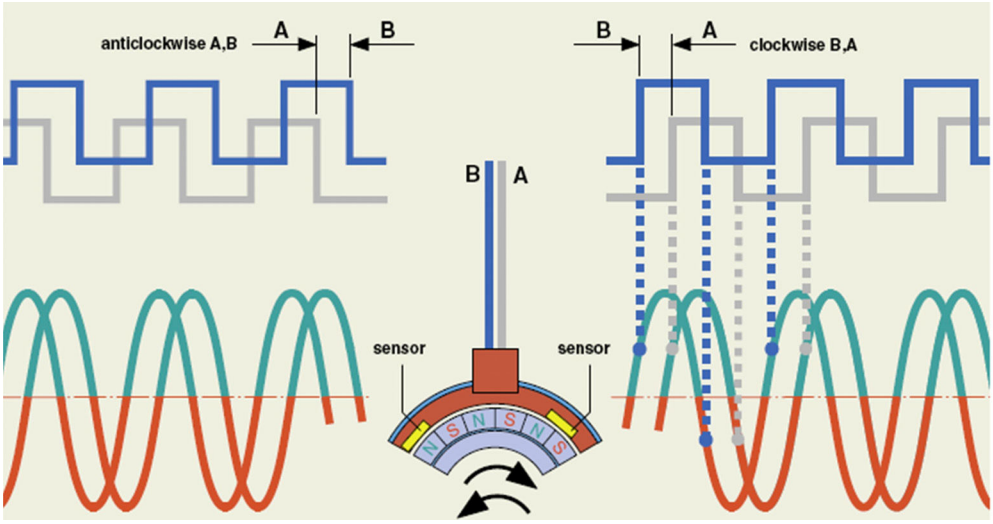

Abbildung 31 – Funktionsweise eines Drehzahl-Sensors [SKF2003] [Haas2007]

Der Ausgang dieses Timers wird mit dem Eingang eines weiteren Hardware-Timers des M32C-Mikrocontrollers verbunden. Mit dem zweiten Timer wird eine sogenannte Intelligent-I/O Funktion realisiert, die die Zeitmessung der Periodendauer ermöglicht, indem der aktuelle Zählerstand des Timer mit jeder steigenden Flanke in einem speziellen Register gespeichert wird. Dieses Register muss in regelmäßigen Abständen von der Anwendung abgefragt und auf einen neuen Messwert überprüft werden. Dabei muss das Abfrageintervall so kalkuliert werden, dass das Register auch bei der maximal möglichen Frequenz ausgelesen wird, bevor der Wert mit der nächsten steigenden Flanke überschrieben wird. Die nachfolgende Berechnung des Abfrageintervalls basiert auf der Annahme, dass die Höchstgeschwindigkeit 400 km/h beträgt und der Radumfang bei 1,5 Meter liegt.

$$\begin{aligned} Zeit\ pro\ Radumdrehung &= \frac{Maximale\ Geschwindigkeit}{Radumfang} \\ &= \frac{400\ km/h}{1{,}5\ m} = \frac{111{,}1\ m/s}{1{,}5\ m} = \underline{\underline{0{,}0135\ s}} \end{aligned} \tag{5.1}$$

$$Maximale\ Pulsfrequenz = \frac{Pulse\ pro\ Umdrehung}{Zeit\ pro\ Umdrehung} = \frac{48}{0{,}0135\ s} \approx \underline{\underline{3360\ Hz}} \tag{5.2}$$

Aus der maximalen Pulsfrequenz der Formel (5.2) ergibt sich, dass alle 280 Mikrosekunden ein neuer Wert im Register des Timers vorliegt, wenn die maximale Geschwindigkeit von 400 km/h zu Grunde gelegt wird. Somit muss für den zu realisierenden Knoten ein Task vorgesehen werden, der mit dem berechneten Intervall von 280 Mikrosekunden aktiviert wird, das Register ausliest und aus der Differenz zweier Messwerte die Radumlaufgeschwindigkeit berechnet.

5.5.3 Gateway

Für die Datenaufzeichnung der Sensorwerte wird ein Netzwerkknoten als Gateway konfiguriert, der die anfallenden Daten aus dem FlexRay-Netzwerk über eine serielle Schnittstelle an einen Rechner weiterleitet.
So ist die Möglichkeit gegeben, die Daten mit Hilfe einer Benutzeroberfläche zu visualisieren und die Datenaufzeichnung aufgrund der höheren Speicherkapazität des Rechners zu realisieren. Durch die im Verhältnis zum FlexRay-Bus deutlich geringere Übertragungsrate der seriellen Schnittstelle RS232 werden ausschließlich Sensorwerte übertragen, bei denen seit dem letzten Sendevorgang Änderungen aufgetreten sind.
Im Gegensatz zum FlexRay-Kommunikationssystem, bei dem die Nachrichten im statischen Segment stets zu festgelegten Zeitpunkten eintreffen und damit auch der Sender und der Inhalt der Nachrichten eindeutig zugewiesen werden kann, muss für die Übertragung per RS232 ein Protokoll festgelegt werden, das die Interpretation der Daten beim Empfänger ermöglicht. Hierfür wird jedem Sensorwert eine Identifikationsnummer zugewiesen und bei der Übertragung per serieller Schnittstelle den eigentlichen Daten vorangestellt, sodass auf dem Rechner die Interpretation anhand der eindeutigen Nummern erfolgen kann.
In Abbildung 32 ist die implementierte Benutzeroberfläche dargestellt, die für die Visualisierung der Sensorwerte eingesetzt wird. Außerdem wird die Eingabe von Daten unterstützt, die über die serielle Schnittstelle zum Gateway-Knoten gesendet werden können und die Konfiguration der Netzwerkknoten zur Laufzeit ermöglicht.

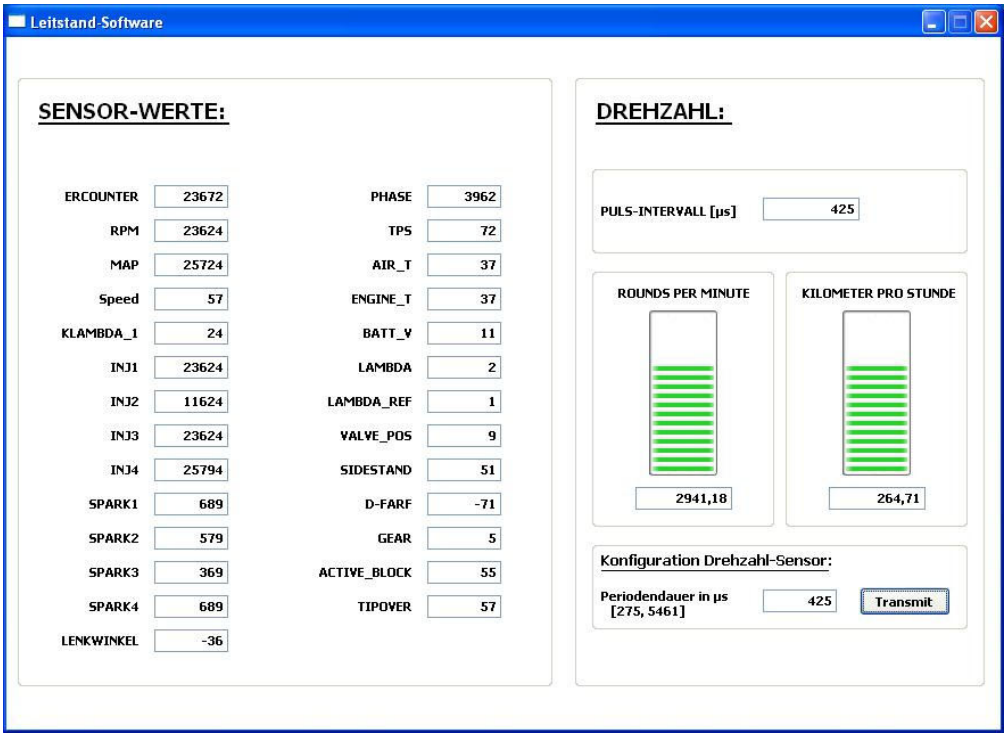

Abbildung 32 – Visualisierung der Sensorwerte

5.5.4 Taskplanung und Entwurf des Applikationszyklus

Mit der Spezifikation der zusätzlichen Aktivitäten, wie zum Beispiel die Softwaremodellierung der Sensoren, und unter Berücksichtigung der Cluster-Spezifikation, können die benötigten Tasks und deren Aktivierungszeitpunkte der Netzwerkknoten im jeweiligen Applikationszyklus festgelegt werden. In Abbildung 33 ist die statische Taskplanung der vier Netzwerkknoten dargestellt, wobei die Länge der Applikationszyklen der Zykluszeit des Kommunikationssystems entspricht. Des Weiteren wird die Zuordnung der Knoten zu den Static-Slots des statischen Segments im FlexRay-Kommunikationszyklus verdeutlicht.

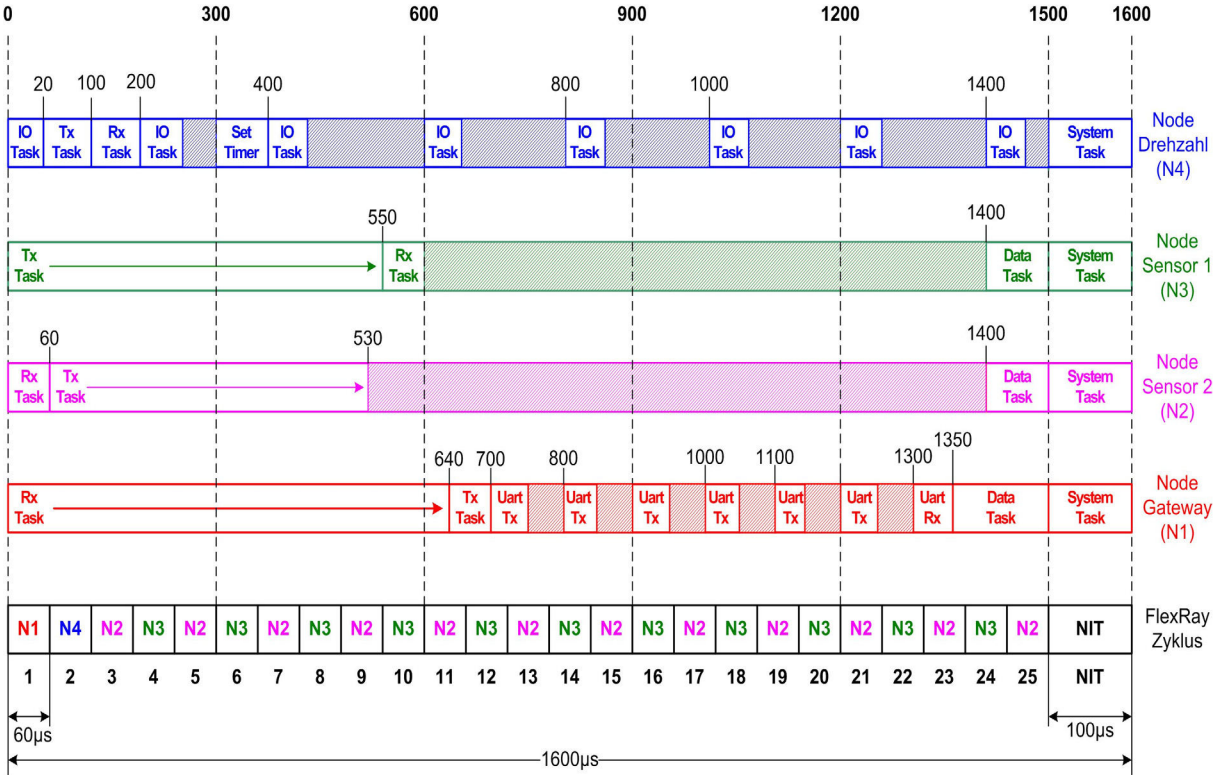

Abbildung 33 – Entwurf der Applikationszyklen

Die damit abgeschlossene Komponentenspezifikation ermöglicht die Konfiguration und die Implementierung der beschriebenen Software-Module (vgl. 5.2), wobei der Commstack-Code und der Betriebsystem-Code durch den abschließenden Einsatz des Werkzeugs DECOSYS:: DESIGNER beziehungsweise ProOSEK-Configurator generiert werden können. Nach der Fertigstellung des Applikationscodes, der die Implementierung der Tasks umfasst und das geplante Verhalten der Komponenten realisiert, können die Knoten getestet und mit der letzten Phase des zweistufigen Entwicklungsansatzes in das System integriert werden.

6. Zusammenfassung

In der vorliegenden Arbeit wird ein verteiltes, eingebettetes Echtzeitsystem auf der Basis eines FlexRay-Kommunikationssystems entwickelt. Der Fokus liegt dabei auf der Migration eines TTCAN-Systems, das für die mikrocontrollerbasierte Telemetrie in einem Formula Student Rennwagen eingesetzt wird, zu einem FlexRay-System.
Durch den Vergleich von FlexRay, mit verschiedenen anderen im Automobilbereich verwendeten Bussystemen, werden die sicherheitsorientierten Eigenschaften verdeutlicht, die für die Entwicklung eines Echtzeitsystems eingesetzt werden können, um auch harten Echtzeitanforderungen bezüglich Sicherheit, Verfügbarkeit und Fehlertoleranz gerecht zu werden. Das unter diesen Gesichtspunkten realisierte System bietet die Option, zukünftige sicherheitsrelevante Steuerungsaufgaben zuverlässig zu bewältigen.

Der Entwurf und die Realisierung basieren auf einem zweistufigen Entwicklungsansatz, nach dem zunächst die Designentscheidungen bezüglich des Gesamtsystems ausgearbeitet werden. Als Ergebnis dieser Phase geht eine Spezifikation hervor, die unter anderem die erforderliche deterministische Kommunikation beschreibt und den zeitgesteuerten Buszugriff in einem TDMA-Ablaufplan zusammenfasst. Auf der Grundlage dieser Spezifikation können im zweiten Schritt des Entwicklungsansatzes die einzelnen Subsysteme unabhängig voneinander entworfen, implementiert und in das Gesamtsystem integriert werden. Durch diese Vorgehensweise wird die komplexe Aufgabe der Koordination der unterschiedlichen Subsystem-Funktionen in einer frühen Entwicklungsphase angegangen und verhindert durch den festgelegten Buszugriff die Überlastung des Kommunikationssystems während der Integration und im Betrieb.

Für die Implementierung der Subsysteme werden zwei verschiedene Echtzeitbetriebssysteme verwendet und eine anschließende Bewertung der beiden Lösungswege auf Grundlage einer messtechnischen Analyse durchgeführt. Die Echtzeitbetriebssysteme werden vorwiegend für die zeitgesteuerte Taskaktivierung eingesetzt, um das spezifizierte Verhalten und die Koordination der einzelnen Subsysteme zu gewährleisten. Für die zeitlich korrekte Ausführung der Tasks müssen die lokalen Uhren der Netzwerkteilnehmer synchronisiert werden. Für die Umsetzung wird ein Task implementiert, der in regelmäßigen Abständen die lokale Uhrzeit an die von Kommunikationssystem etablierte, globale Zeit anpasst.

Für den Versuchsaufbau im Labor werden die realen Verhältnisse durch die Implementierung von Software-Komponenten und durch die Konfiguration der Hardware auf den eingesetzten FlexRay-Knoten nachgebildet. Das resultierende Design des Applikationszyklus ist auf die Simulation der Sensoren ausgelegt und berücksichtigt die Anforderung, die anfallenden Daten zu erfassen und eine höchstmögliche Auslastung des Kommunikationssystems zu erreichen.

6.1 Ausblick und Möglichkeiten

Bei der Konstruktion eines Formula Student Rennwagens ist es erforderlich, dass die verwendeten Komponenten ein möglichst geringes Gewicht ausweisen und die Kosten bei der Finanzplanung berücksichtigt werden. Da die FlexRay-Technologie in der Automobilindustrie bereits in die Serienproduktion gebracht worden ist und immer mehr Halbleiterhersteller Mikrocontroller mit integriertem FlexRay-Controller anbieten, wird der finanzielle Aspekt bei dem Aufbau eines FlexRay-basierten Netzwerks stetig in den Hintergrund rücken.
Für die zusätzlich erforderliche On-Board-Aufzeichnung der Bus-Kommunikation werden ebenfalls Entwicklungen angeboten, die die Umgebungsanforderungen für den Einsatz in einem Fahrzeug berücksichtigen und somit auch die Randbedingungen für den Einsatz in einem Formula Student Rennwagen, wie Temperaturbeständigkeit und Beständigkeit gegenüber Wasser und Erschütterungen, erfüllen können. Durch die direkte Verbindung zum Bussystem ist bei einer On-Board-Aufzeichnung gewährleistet, dass der gesamte Datenverkehr verlustfrei gesichert werden kann und eine spätere Auswertung der Daten möglich ist.

Des Weiteren ist es für die Integration des Systems in einen Rennwagen notwendig, eine Funkverbindung für die Datenübertragung aus dem FlexRay-Netzwerk zu einem Leitstand am Streckenrand herzustellen. So können das Fahrzeug und das System im laufenden Betrieb kontrolliert werden und der Fahrer im Falle einer kritischen Situation oder einer Fehlfunktion unterrichtet werden.
Für die Umsetzung könnte die Verwendbarkeit des im TTCAN-System eingesetzten WLAN-Moduls im Umfeld eines FlexRay-Netzwerks geprüft und gegebenenfalls andere Hardware-Lösungen untersucht werden und so die Entwicklung des bestehendes Projektes fortzuführen.

Durch die Verwendung eines FlexRay-Kommunikationssystems ist die Basis für die Realisierung von sicherheitskritischen Steuerungen gelegt, die die Handhabung des Fahrzeugs weiter verbessern und den Fahrer in schwierigen Situationen zuverlässig unterstützen können. Dabei kann der Ansatz für den Aufbau des Kommunikationszyklus analog zu dem migrierten TTCAN-System erfolgen, bei dem Zeitfenster für die nachträgliche Integration von zusätzlichen Komponenten freigehalten worden sind. Dadurch ist die Möglichkeit gegeben, die TDMA-Planung anzupassen und anschließend lediglich bei den Subsystemen Software-Änderungen vorzunehmen, beziehungsweise den aktualisierten TDMA-Ablaufplan bereitzustellen, die von den Änderungen betroffen sind.
Neben der bereits geplanten elektronischen Stabilisierung des Fahrzeugs ist auch die Realisierung eines Antiblockiersystems denkbar.

7. Literaturverzeichnis

[Raus2008] Rausch, Mathias: *FlexRay: Grundlagen, Funktionsweise, Anwendung*. Hanser Fachbuchverlag, 2008. ISBN: 3-446-41249-2

[Form2008] Formula Student Germany, 2008 – URL http://www.formulastudent.de/de/ueber-formula-student-germany/konzept/, zugegriffen am 05.08.2008

[Schu2007] Schuckert, Simon: *Mikrocontrollerbasierte Telemetrie und Echtzeitauswertung von Sensordaten im Formula Student Rennwagen*. Hochschule für Angewandte Wissenschaften Hamburg, Bachelorarbeit, 2007

[Haas2007] Haase, Sebastian: *Telemetrie im Formula Student Rennwagen auf Basis von CAN Bus, Datenspeicherung und Wireless LAN Technologien*. Hochschule für Angewandte Wissenschaften Hamburg, Bachelorarbeit, 2007

[Sell2006] Sellentin, Jörn: *Ein zeitgesteuertes, verteiltes SW-Konzept implementiert auf FlexRay-Komponenten für ein fahrerloses Transportsystem*. Hochschule für Angewandte Wissenschaften Hamburg, Masterarbeit, 2006

[Wiew2004] Wiewesiek, Wolfgang: *Entwicklungsplattform für FlexRay Anwendungen*. Fujitsu Microelectronics, 2006

[Schu2006] Schuermans, Roel: *Durchgängige Entwicklung von FlexRay-Systemen*. Automobil-Elektronik, Juni 2006

[Geva2005] Gevatter, Hans-Jürgen; Grünhaupt, Ulrich: *Handbuch der Mess- und Automatisierungstechnik im Automobil*. Springer-Verlag, 2005. ISBN: 3-540-21205-1

[Etsc2002] Etschberger, Konrad: *Controller Area Network*. Hanser Fachbuchverlag, 2002. ISBN: 3-446-21776-2

[Knol2006] Knoll, Alois: *Echtzeitfähige Kommunikation: Medienzugriffsverfahren*. Technische Universität München, 2006

[Pree2007] Pree, Wolfgang: *Softwareentwicklung für verteilte Systeme im Automobil*. Universität Salzburg, 2007

[Reif2007] Reif, Konrad: *Automobilelektronik – Eine Einführung für Ingenieure*. Vieweg-Verlag, 2007. ISBN: 978-3-8348-0297-2

[Bosc2000] Führer, Thomas; Müller, Bernd: *Time Triggered Communication on CAN*. Robert Bosch GmbH, 2000

[LIN2006] LIN-Konsortium: *LIN Specification Package*. Revision 2.1, 2006

[Grie2000] Grießbach, Robert; Berwanger, Josef; Peller, Martin: *Byteflight – neues Hochleistungs-Datenbussystem für sicherheitsrelevante Anwendungen*. ATZ/MTZ Automotive Electronics, 2000

[Szec2007] Szecówka, P.M.; Świderski, M.A.: *On Hardware Implementation of FlexRay Bus Guardian Module*. Wroclaw University of Technology Poland, 2007

[Temp1998] Temple, Christopher: *Avoiding the Babbling-Idiot Failure in a Time-Triggered Communication System*. 1998

[Temp2003] Temple, Christopher: *FlexRay – Protocol Overview*. FlexRay International Workshop, 2003

[Will2006] Wille, Marcel: *Bussysteme, FlexRay-Update*. 2006 – URL http://www.elektronikpraxis.vogel.de/themen/hardwareentwicklung/datenkommunikationsics/articles/48502/, zugegriffen am 21.08.2008

[Flex2004] FlexRay Consortium: *FlexRay Communications System Protocol Specification*. FlexRay Consortium, Version 2.1, 2004

[Mill2005] Millsap, Arnie: FlexRay – *Protocol Overview and Development Status*. Vector Congress, 2005

[Weic2005] Weich, Carsten; Plankensteiner, Markus: *Zeitgesteuerte Architektur - Entwicklungsplattform für verteilte Echtzeitsysteme auf FlexRay-Basis*. ElektronikPraxis, 2005

[Tech2007] TTTech Automotive GmbH: *Die zeitgesteuerte Architektur und FlexRay*. 2007 – URL http://www.tttech-automotive.de/technologie/ansatz.htm, zugegriffen am 05.08.2008

[Sche2007] Schedl, Anton: *Goals and Architecture of FlexRay at BMW*. Vector FlexRay Symposium, 2007

[Elen2004] Elend, Bernd: *Design von FlexRay-Netzwerk-Topologien für verteilte sicherheitsrelevante Applikationen*. Auto & Elektronik, 2004

[Homa2004] Homann, Matthias: *OSEK: Betriebssystem-Standard für Automotive und Embedded Systems*. Mitp-Verlag, 2004. ISBN: 3-826-61552-2

[Ring2002] Ringler, Thomas: *Entwicklung und Analyse zeitgesteuerter Systeme*. Universität Stuttgart, 2002.

[Scho2001] Schoof, Jochen: *OSEKtime – Standard für zeitgesteuerte Betriebssysteme*. 3SOFT GmbH Erlangen, 2001.

[OSEK2001] OSEK/VDX: *Time-Triggered Operating System Specification*. Version 1.0, 2001.

[De1_2006] DECOMSYS: *DECOMSYS::NODE<RENESAS> User Manual*. Dependable Computer Systems GmbH, Version 1.3, 2006

[De2_2006] DECOMSYS: *APPLICATION EXECUTION SYSTEM User Manual*. Dependable Computer Systems GmbH, Version 1.1, 2006

[De3_2006] DECOMSYS: *COMMSTACK<FLEXRAY> User Manual*. Dependable Computer Systems GmbH, Version 1.8, 2006

[De4_2006] DECOMSYS: *FLEXRAY SYNC HANDLER Technical Reference Manual*. Dependable Computer Systems GmbH, Version 1.2, 2006

[De5_2006] DECOMSYS: *OS SYNCHRONIZATION HANDLER Technical Reference Manual*. Dependable Computer Systems GmbH, Version 1.1, 2006

[SKF2003] SKF: *Sensor-Bearing Units concentrate intelligence in your motion control*. 2003 – URL http://www.skf.com, zugegriffen am 05.08.2008

[Pasc2007] Pascoli, Alessandro: *Diagnostic Structure and End Line Parameters for ECUA firmware application*. Version 0.8, 2007

8. Abbildungsverzeichnis

9. Glossar

AES	Application Execution System, zeitgesteuertes Betriebssystem für eingebettete Systeme
ANSI-C	American National Standards Institute, Norm, die eine allgemeingültige Definition der Syntax für die Programmiersprache C beschreibt.
API	Application Programming Interface, Programmierschnittstelle für den Zugriff auf Datenbanken und Hardware oder das Erstellen einer grafischen Benutzeroberfläche
Bus Guardian	Vorrichtung zur Kontrolle des Buszugriffs bei einem TDMA-Verfahren
byteflight	Bussystem, das für sicherheitskritische Anwendungen im Automobilbereich entwickelt worden ist
CAN	Controller Area Network, Standardisierter, von Bosch entwickelter, Datenbus
CHI	Controller-Host-Interface, Schnittstelle zwischen Mikrocontroller und Kommunikationscontroller
Coldstarter	Speziell konfigurierter Knoten eines FlexRay-System, der für die Startphase und die Integration der Teilnehmer ins Netzwerk zuständig ist
Commstack	Software-Bibliothek, die eine Abstraktionsschicht für den Zugriff auf einen Kommunikationscontroller darstellt
CRC	Cyclic Redundancy Check, Algorithmus zur Fehlererkennung bei Datenübertragungen mittels Berechnung von Prüfsummen
CSMA/CA	Carrier Sense Multiple Access/Collision Avoidance, zufälliges Buszugriffsverfahren bei dem Kollisionen, die durch den gleichzeitigen Zugriff entstehen, vermieden werden
CSMA/CD	Carrier Sense Multiple Access/Collision Detect, zufälliges Buszugriffsverfahren bei dem Kollisionen, die durch den gleichzeitigen Zugriff entstehen erkannt werden
Dispatcher	Teil eines Betriebssystems, der die zur Verfügung stehende Prozessorzeit auf die anstehenden Aufgaben verteilt

Frame	Bezeichnung einer Nachricht in einem FlexRay-Kommunikationssystem
FTDMA	Flexible Time Division Multiple Access, ein auf TDMA basierendes, zeitgesteuertes Buszugriffsverfahren bei dem die Zeitschlitze eine variabler Länge besitzen
Gateway	Verbindungsglied für die Kommunikation zwischen Netzwerken, die auf unterschiedlichen Protokollen basieren
JTAG	Joint Test Action Group, ein Verfahren zum Testen und Debuggen von elektronischer Hardware
LIN	Local Interconnect Network, Kommunikationssystem mit geringen Datenübertragungsraten für den Einsatz in Kraftfahrzeugen.
Macrotick	Globale Zeiteinheit in einem FlexRay-Kommunikationssystem
Microtick	Lokale Zeiteinheit in einem FlexRay-Kommunikationssystem
Minislot	Zeitschlitz von variabler Länge im FlexRay-FTDMA-Verfahren
MTS	Media Test Symbol, Kommunikationselement in einem FlexRay-System für den Test eines Bus Guardians
NIT	Network Idle Time, kommunikationsfreier Abschnitt eines FlexRay-Kommunikationszyklus
NRZ	Non Return to Zero, Codierungsverfahren für die Übertragung von Datenbits auf Busleitungen
OIL	OSEK Implementation Language, Beschreibungssprache für OSEK/VDX-Systeme
OSEK/VDX	Offene Systeme und deren Schnittstellen für die Elektronik im Kraftfahrzeug/Vehicle Distributed Executive, ein industrielles Standardisierungsgremium, das von mehreren Kfz-Herstellern gegründet wurde
OSEKtime	standardisiertes Echtzeitbetriebssystem des OSEK/VDX-Konsortiums für eingebettete Systeme
Payload	Anteil einer Nachricht, der für die Übertragung der eigentlichen Nutzdaten vorgesehen ist
Static-Slot	Zeitschlitz von fester Länge im FlexRay-TDMA-Verfahren

TDMA	Time Division Multiple Access, zeitgesteuertes Buszugriffsverfahren, bei dem Zeitschlitze gleicher Länge festgelegt werden, zu denen ein Teilnehmer den exklusiven Buszugriff erhält
TTCAN	Time-Triggered CAN – Protokoll das auf dem CAN-Bus aufsetzt und eine Echtzeitsteuerung ermöglicht
WLAN	Wireless Local Area Network, lokales Funknetz mit kurzer Reichweite
X-by-Wire	Realisierung von Funktionen durch mechatronische Lösungen

A. Installation und Verwendung der Software für den Aufbau einer FlexRay-Strecke mit dem Decomsys StarterKit und Netzwerkknoten vom Typ Decomsys Node<Renesas>

A.1 Überblick Hardware und Software-Komponenten

Als Einleitung wird in diesem Kapitel ein Überblick über die verwendeten Komponenten für den Aufbau einer FlexRay-Strecke dargestellt. In Kapitel A.1.1 wird die benötigte Hardware und in Kapitel A.1.2 die Software vorgestellt, mit der ein Netzwerk erstellt werden kann, bei dem die Kommunikation der Netzwerkknoten über einen FlexRay-Bus realisiert werden soll.

A.1.1 Hardware

In Abbildung 34 ist ein exemplarischer Labor-Aufbau der verwendeten Hardware dargestellt. Hierbei handelt es sich um zwei Netzwerkknoten der Firma Decomsys vom Typ Node<Renesas>. Ein Knoten wird im Folgenden auch als Electronic Control Unit (ECU) bezeichnet. Eine ECU besteht aus einem Mikrocontroller, auf dem die eigentliche Anwendung läuft und einem Kommunikationscontroller, der für das Senden und Empfangen der Frames auf dem FlexRay-Bus zuständig ist. Zur Ausstattung der Knoten gehören zwei CAN Kanäle, eine LIN Schnittstelle, eine RS232 Schnittstelle sowie eine Reihe digitaler und analoger Ein- und Ausgänge. Zu den erforderlichen Schnittstellen für die Realisierung des hier beschriebenen Netzwerks zählen die Anschlüsse der beiden FlexRay-Kanäle und ein Debug-Anschluss, über den die Software auf die Knoten geladen wird. Die Verbindung vom Debug-Anschluss zu einem USB-Anschluss eines Rechners, auf dem die Entwicklungsumgebung läuft, wird durch den RTA-FoUSB-MON Flash Programmer & In-Circuit Debugger der Firma Renesas hergestellt, der in der Abbildung als USB-Programmer bezeichnet wird.

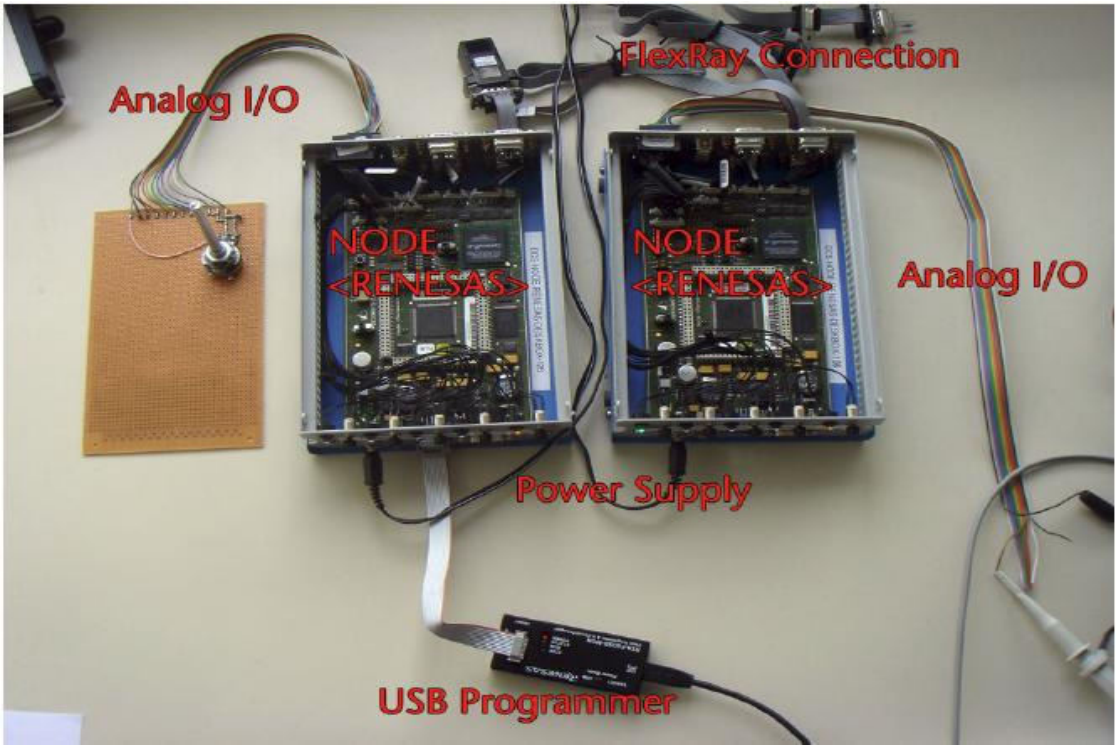

Abbildung 34 – Hardware-Aufbau

A.1.2 Software

Die verwendete Software besteht im Wesentlichen aus drei verschiedenen Tools:

- DECOMSYS::DESIGNER
- IAR Embedded Workbench IDE
- KD3083-Debugger

Mit dem DESIGNER-Werkzeug der Firma Decomsys wird die Konfiguration, die den FlexRay-Bus und das FlexRay-Protokoll betrifft, erstellt (vgl. Abbildung 35).
Als zweites Werkzeug wird die in Abbildung 36 dargestellte IAR Embedded Workbench verwendet, womit die eigentlichen Applikationen der Knoten implementiert und der automatisch generierte C-Code des Designer-Tools eingebunden wird. Mit dem Compiler und Linker der IAR Workbench wird der Objekt-Code generiert und durch die Verwendung des dritten Tools auf den jeweiligen Knoten geladen.
Bei diesem dritten Werkzeug handelt es sich um den KD3083-Debugger (vgl. Abbildung 37). Hiermit kann das fertig gestellte Programm auf den jeweiligen Knoten geladen und auch im Debug-Modus ausgeführt werden.

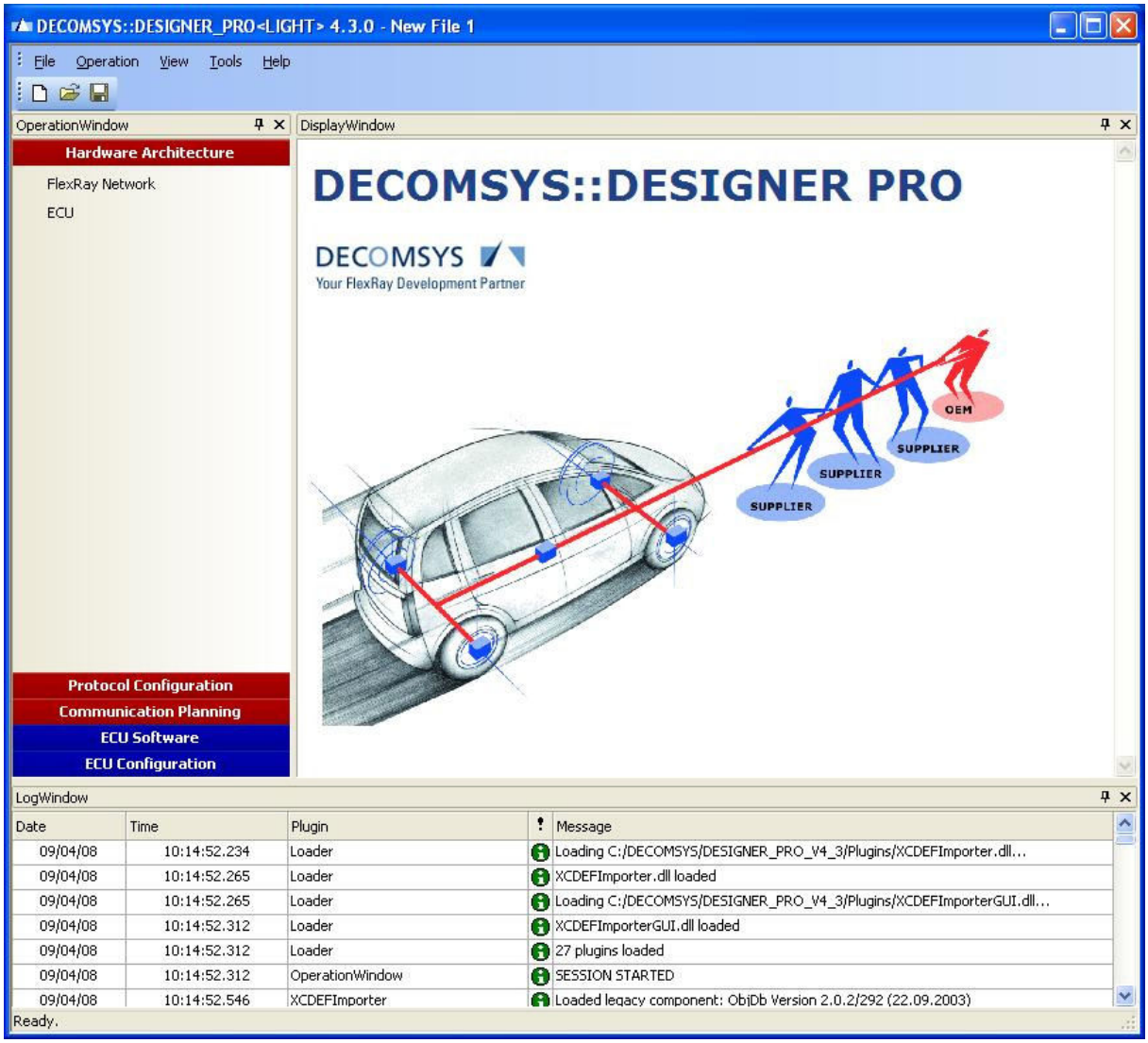

Abbildung 35 – DECOMSYS::DESIGNER

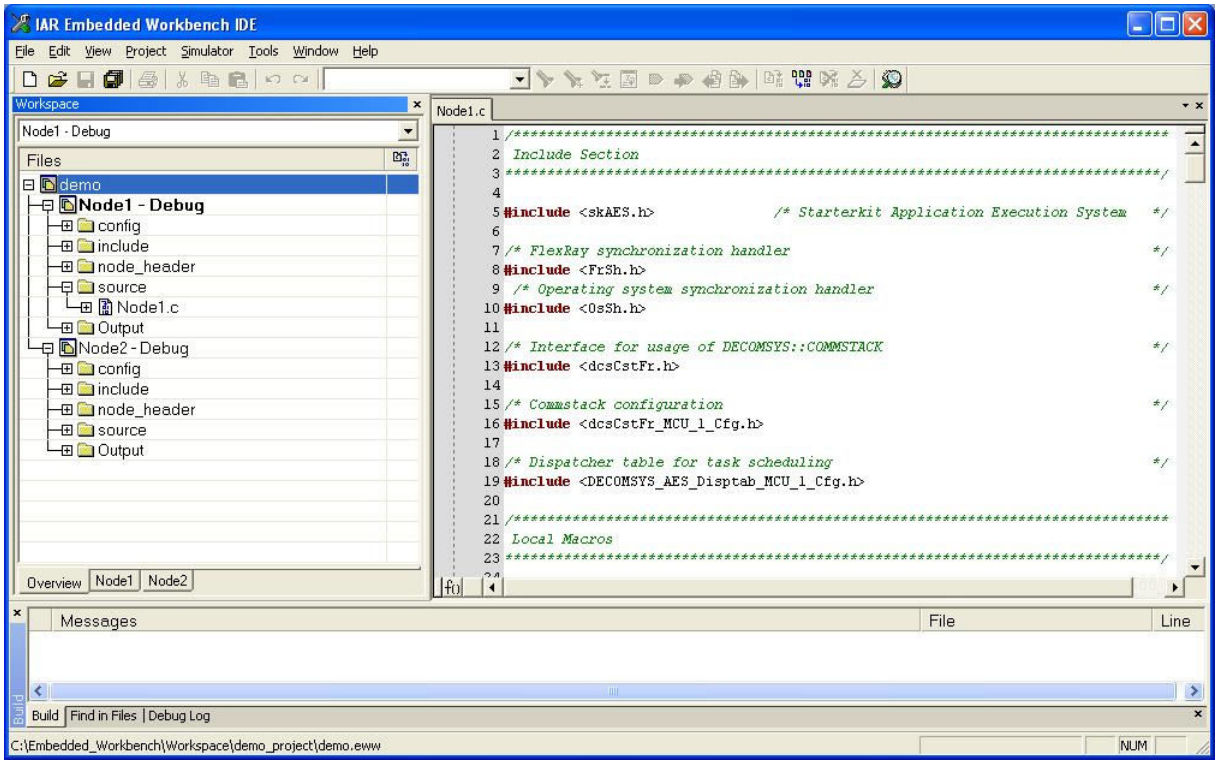

Abbildung 36 – IAR Embedded Workbench

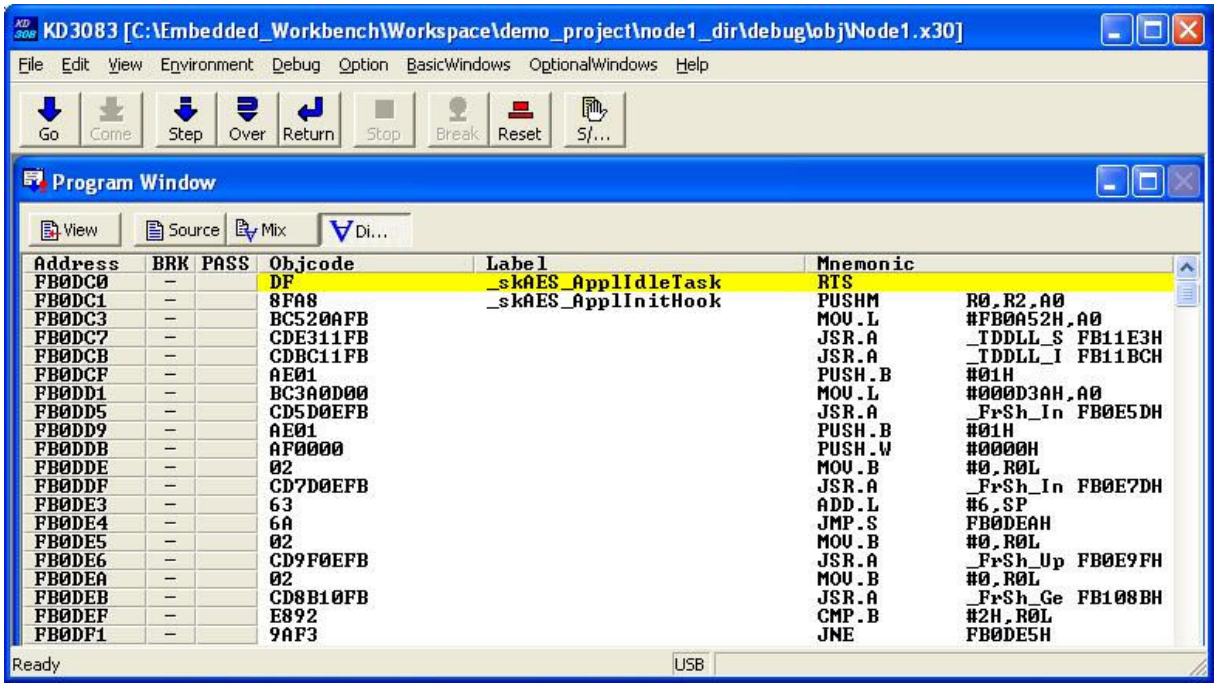

Abbildung 37 – User-Interface des KD3083-Debuggers

A.2 Software-Installation

A.2.1 DECOMSYS::DESIGNER Pro 4.3

Nach der Installation des DECOMSYS::DESIGNER Werkzeugs ist die Installation des Commstack-Configurator erforderlich. Dieses Plug-In stellt eine Abstraktionsschicht für die Schnittstelle zwischen Mikrocontroller und FlexRay-Kommunikationscontroller dar und ermöglicht die frame-basierte Kommunikation über den FlexRay-Bus.
Für die Verwendung der Designer-Software ist neben einer Lizenz-Datei noch ein USB-Dongle erforderlich. Dieser schaltet, in dem an der HAW vorliegenden Lizenz-Modell, einen einzelnen Rechner für die Benutzung der Software frei. Für den USB-Dongle ist eine Treiber-Installation notwendig, die mit dem Tool FLEXid-Installer durchgeführt werden kann. Die aktuelle Version ist über den folgenden Link erhältlich:
http://www.aladdin.com/support/hasp/hasp4/enduser.aspx

A.2.2 IAR Embedded Workbench for Renesas M32C V3

Bei der Installation der IAR Embedded Workbench ist zu beachten, dass ein Verzeichnispfad ohne Leerzeichen im Namen angegeben wird, da ansonsten Probleme bei der Kompilierung des Quellcodes auftreten können.

A.2.3 DECOMSYS::Node<Renesas> StarterKit Version 2.0

Durch die Installation des StarterKits werden hilfreiche Software-Bibliotheken bereitgestellt, die unter anderem den Zugriff auf den FlexRay-Kommunikationscontroller und die Verwendung des Decomsys Application Execution System (AES) ermöglichen.
Das Echtzeitbetriebssystem AES [De2_2006] stellt einen Dispatcher für die periodische, zeitgesteuerte Taskaktivierung zur Verfügung und unterstützt die Synchronisation der lokalen Uhr des ausführenden Mikrocontrollers unter der Verwendung zweier weiterer Bibliotheken. Dabei handelt es sich um den FlexRay-Sync-Handler [De4_2006], der für die Initialisierung und Konfiguration des Kommunikationscontrollers eingesetzt wird und den OS-Synchronization-Handler [De5_2006], der verwendet wird, um das AES-Betriebssystem auf der Zeit-Basis des laufenden FlexRay-Netzwerks zu synchronisieren.

Außerdem werden drei Beispiele bereitgestellt [De1_2006], die jeweils die Projekt-Dateien des DECOMSYS::DESIGNER Werkzeugs und der IAR Embedded Workbench enthalten. Diese Projekte können mit dem jeweiligen Tool geöffnet werden, sodass die Implementierung beziehungsweise die FlexRay-Konfiguration einsehbar ist und für erste Versuche verwendet werden kann.

A.2.4 KD3083-Debugger

Für den Einsatz des in Abbildung 38 dargestellten RTA-FoUSB-MON Flash Programmers ist zum einen die Installation der Software KD3083-Debugger notwendig und zum anderen ein Treiber für die Hardware. Die Installation des Treibers erfolgt durch das Tool Flash-Over-USB-Programmer, das im Download-Bereich der Firma Renesas unter www.**renesas**.com erhältlich ist.
Vor der Verwendung des RTA-FoUSB-MON Flash Programmers ist sicherzustellen, dass der Power Mode"-Schalter auf "TARGET" eingestellt ist, bevor die Verbindung zum Decomsys Node<Renesas> hergestellt wird, da die Hardware ansonsten durch Überspannung beschädigt werden könnte.

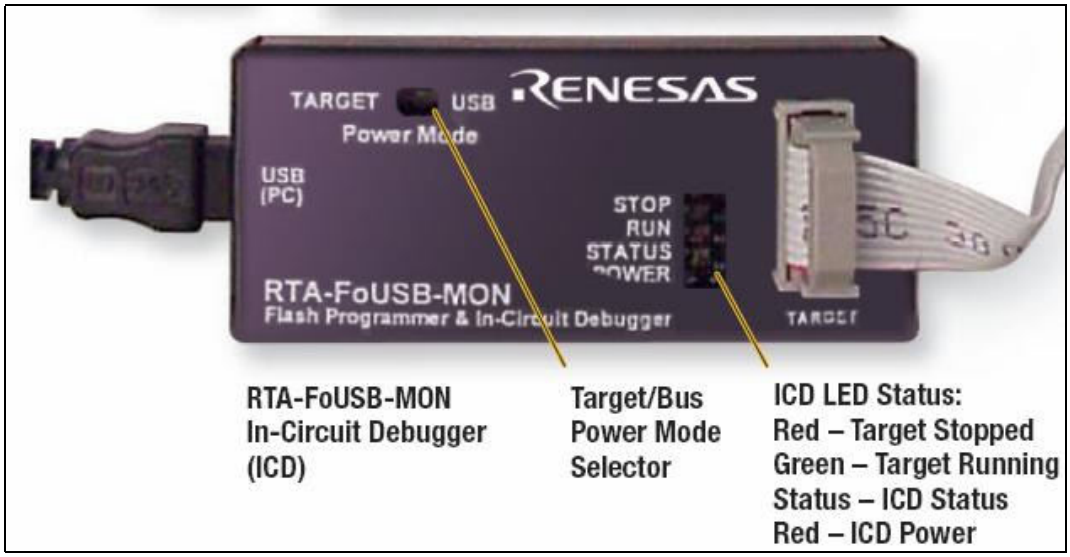

Abbildung 38 – RTA-FoUSB-MON Flash Programmer

A.3 Software-Design

Nach der Installation der benötigten Software kann mit dem Design der Anwendung begonnen werden. Als erstes wird unter Verwendung des DECOMSYS::DESIGNER Werkzeugs das gewünschte Netzwerk erstellt. Ein exemplarischer Aufbau eines solchen Netzwerks ist in Abbildung 39 dargestellt, wobei der abgebildete *Busdoctor* dazu dient, den Datenverkehr auf den FlexRay-Kanälen mit Hilfe der zugehörigen Software zu analysieren.
In den folgenden Abschnitten werden die erforderlichen Arbeitsschritte aufgezeigt und beschrieben, welche Parameter anzugeben sind, um abschließend den entsprechenden, vom Werkzeug generierten C-Code zu erhalten.

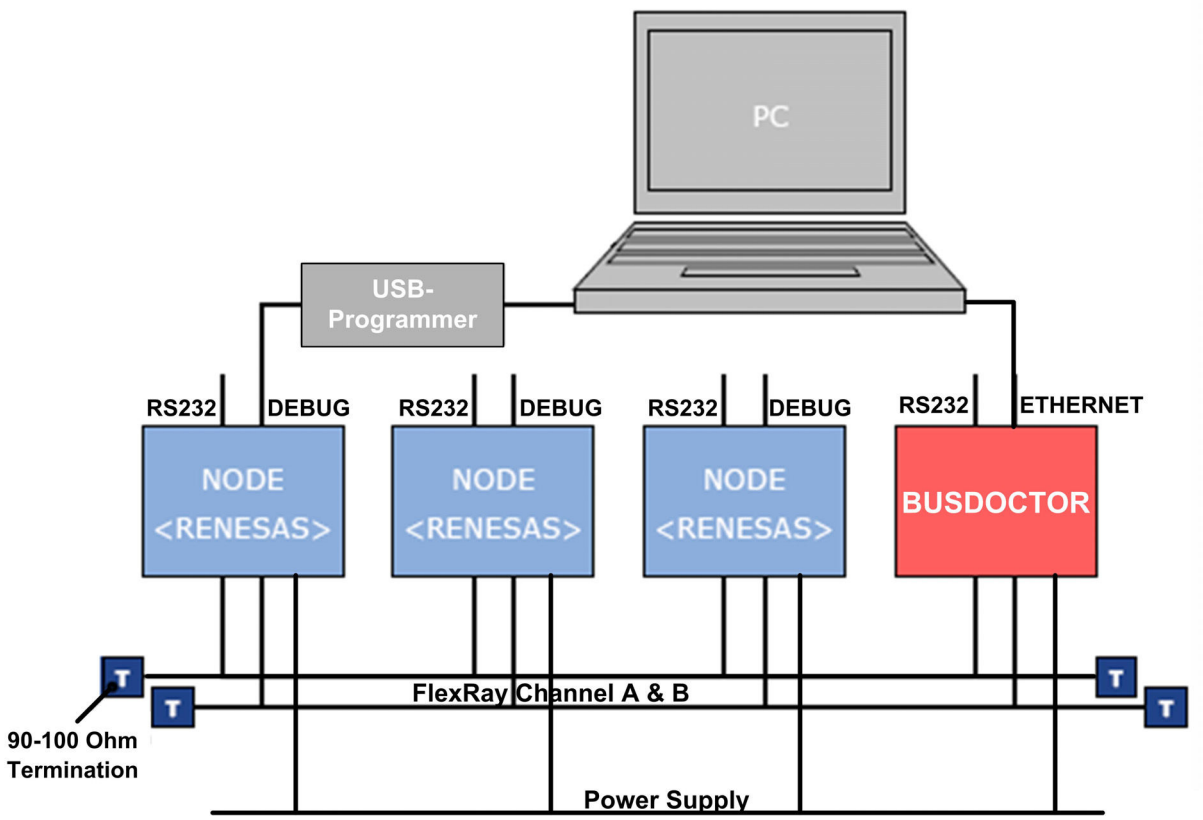

Abbildung 39 – Exemplarischer Aufbau eines Netzwerks

A.3.1 DECOMSYS::DESIGNER PRO 4.3

In Abbildung 40 ist der Startbildschirm des Werkzeugs DECOMSYS::DESIGNER dargestellt. Auf der linken Seite im *Operation-Window* stehen folgende Einträge zur Verfügung:

- *Hardware Architecture*
- *Protocol Configuration*
- *Communication Planning*
- *ECU-Software*
- *ECU-Configuration*

Diese Einträge stehen für die einzelnen Arbeitsschritte, die auszuführen sind, um von einem neuen Projekt zu einem fertigen FlexRay-Design zu gelangen.
Rechts daneben befindet sich das sogenannte *Display-Window*, das die jeweiligen Arbeitsschritte mit Erläuterungen und Grafiken zusätzlich unterstützt.

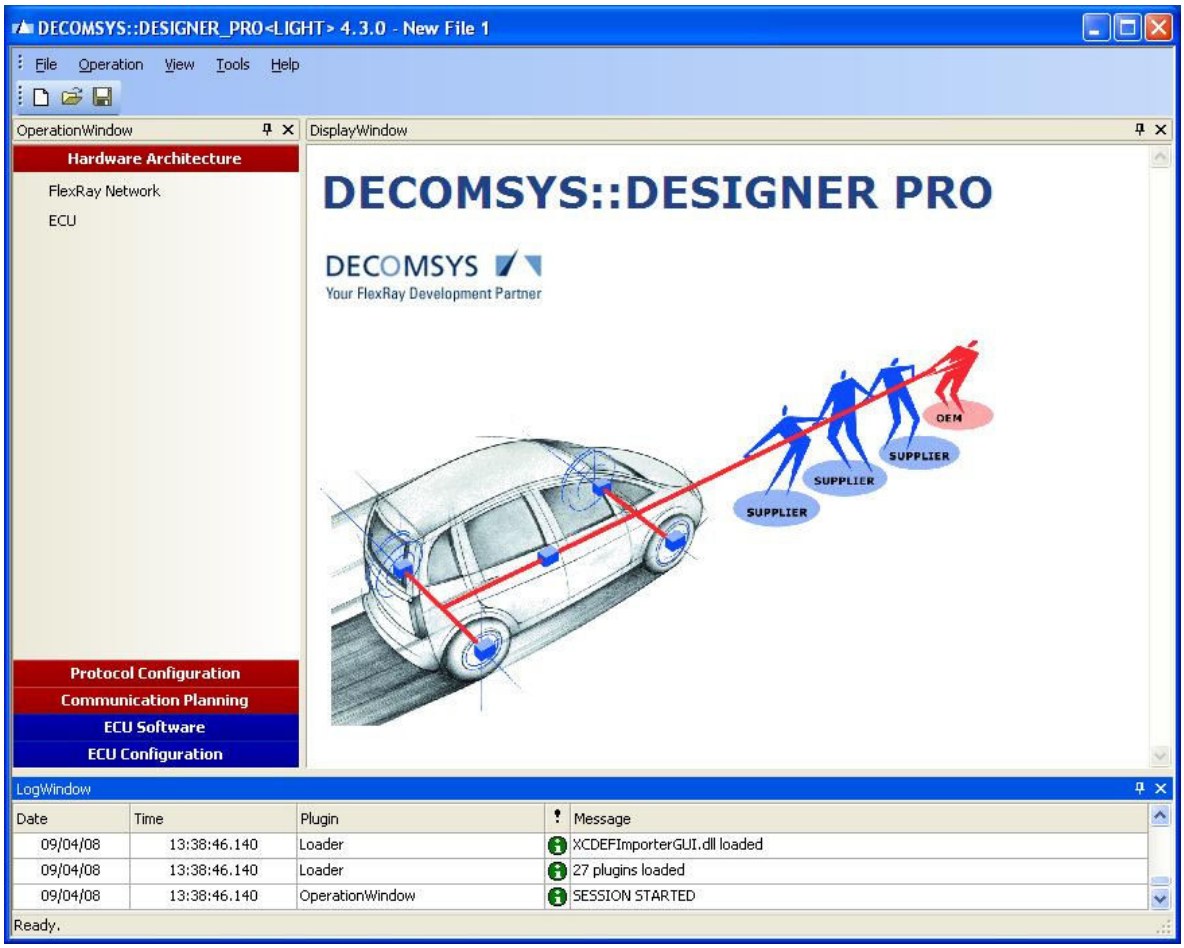

Abbildung 40 - Startbildschirm des DECOMSYS::DESIGNER

A.3.1.1 Hardware Architecture - FlexRay Network

Unter *Hardware Architecture* wird zunächst der Eintrag *FlexRay Network* ausgewählt (vgl. Abbildung 41). Hier wird mit “Create“ ein neues Netzwerk erstellt, was als Vergleich mit anderen Entwicklungsumgebungen der Erstellung eines neuen Projekts entspricht. Im rechten Teil dieses Fensters besteht die Möglichkeit dem Netzwerk einen anderen Namen zuzuweisen und eine Beschreibung hinzuzufügen.
Mit dem Eintrag *Communication Channel* kann festgelegt werden, ob das Netzwerk für den Einsatz mit einem oder zwei FlexRay-Kanälen konfiguriert werden soll.
Anschließend kann das Fenster über “Close“ verlassen werden.

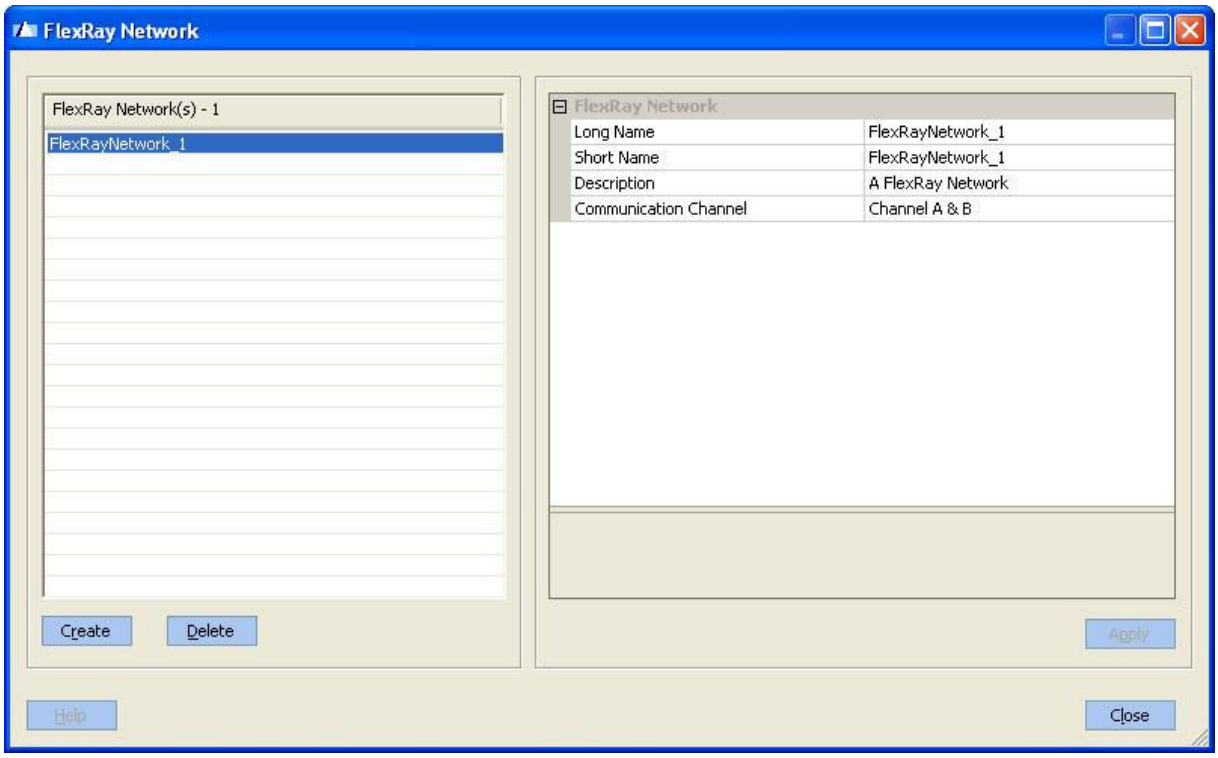

Abbildung 41 - FlexRay Network

A.3.1.2 Hardware Architecture - Electronic Control Unit

Mit dem zweiten Menüpunkt *ECU* dieses Arbeitsschrittes wird die Anzahl der miteinander kommunizierenden Knoten des Netzwerks festgelegt (vgl. Abbildung 42). Eine ECU setzt sich aus einer Mikrocontroller Unit (MCU) und einem Kommunikationscontroller (CC) zusammen. Für das in Abbildung 39 dargestellte Netzwerk, in dem die Node<Renesas>-Blöcke jeweils einer ECU entsprechen, müssen in diesem Fenster dementsprechend 3 ECUs kreiert werden. Für jeden erstellten Knoten müssen anschließend die folgenden Einstellungen vorgenommen werden:

- MCU-Typ → NODE<RENESAS>
- CC-Typ → Renesas: M32C e-ray Beta FPGA CC

Zusätzlich können hier die default-mäßig eingestellten Namen und die *Communication Channel* der Knoten geändert werden. Nach dem Schließen dieses Fensters ist der Arbeitsschritt *Hardware Architecture* abgeschlossen.

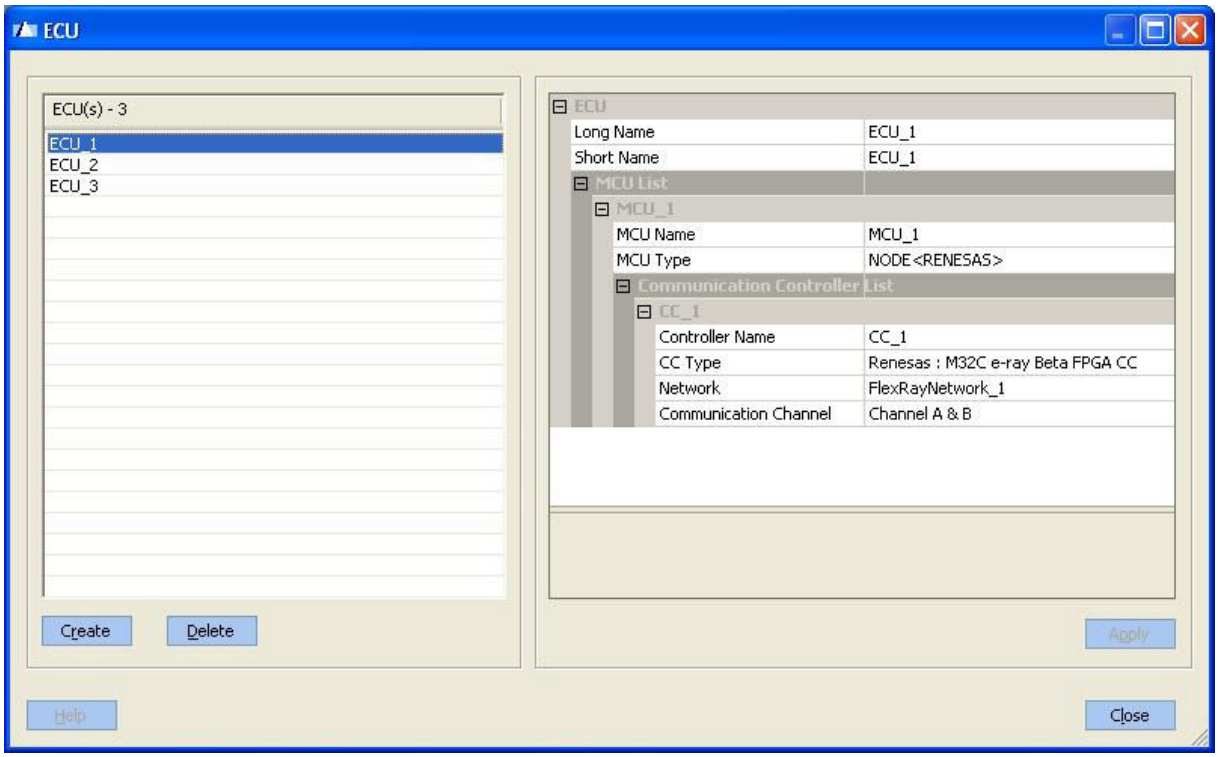

Abbildung 42 - Electronic Control Unit

A.3.1.3 Protokoll Konfiguration

Als zweiter Arbeitsschritt folgt die *Protocol Configuration*. Hier können sämtliche Parameter, die das FlexRay-Protokoll betreffen, definiert werden. Für die ersten Projekte ist die Verwendung des ersten Menüpunkts, des *FlexRay Configuration Wizards*, ratsam. Hier werden die grundlegenden Parameter, die sogenannten High-Level-Parameter, durch den Anwender definiert und alle anderen, der mehr als 70 konfigurierbaren Parameter durch das Werkzeug berechnet beziehungsweise mit Default-Werten belegt.
Erfahrende Anwender können die Feineinstellungen des FlexRay-Clusters mit Hilfe des *FlexRay Configuration Editor* auch manuell vornehmen.

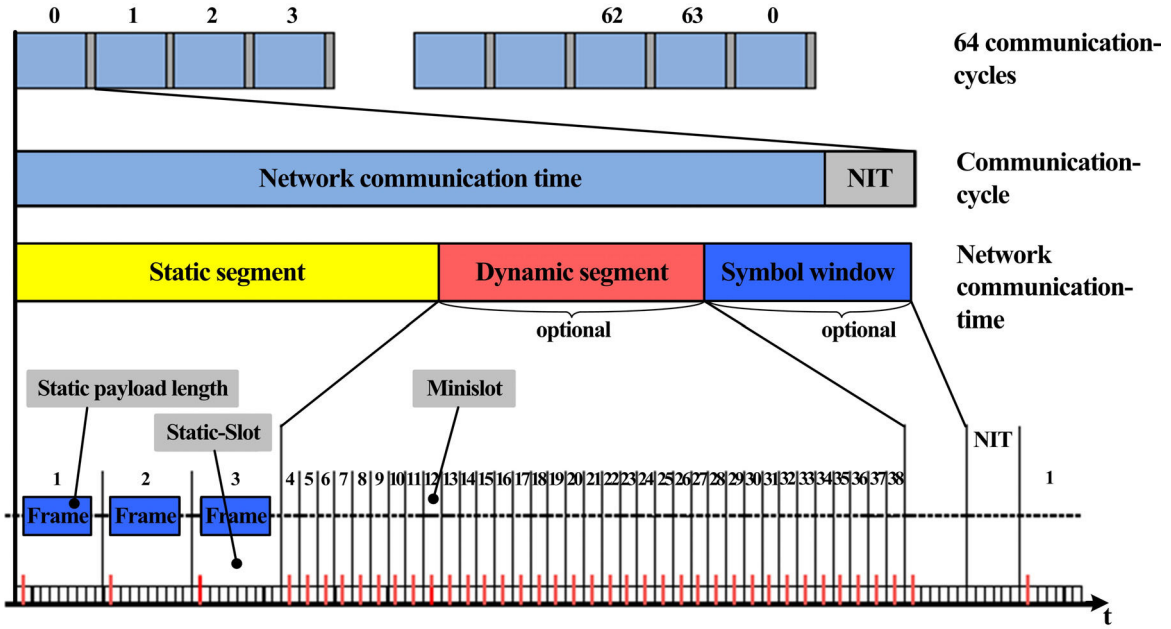

Abbildung 43 – FlexRay Protokoll

FlexRay Configuration
Im oberen Teil des Dialog-Fensters *FlexRay Configuration Wizard* (vgl. Abbildung 44) können die folgenden High-Level-Parameter für den Cluster definiert werden:

- Zyklus-Länge in Macroticks
- Anzahl der Static-Slots
- Länge eines Static-Slots in Macroticks
- Payload-Länge eines Static-Frames
- Länge eines Mini-Slots in Macroticks
- Länge der Network Idle Time in Macroticks

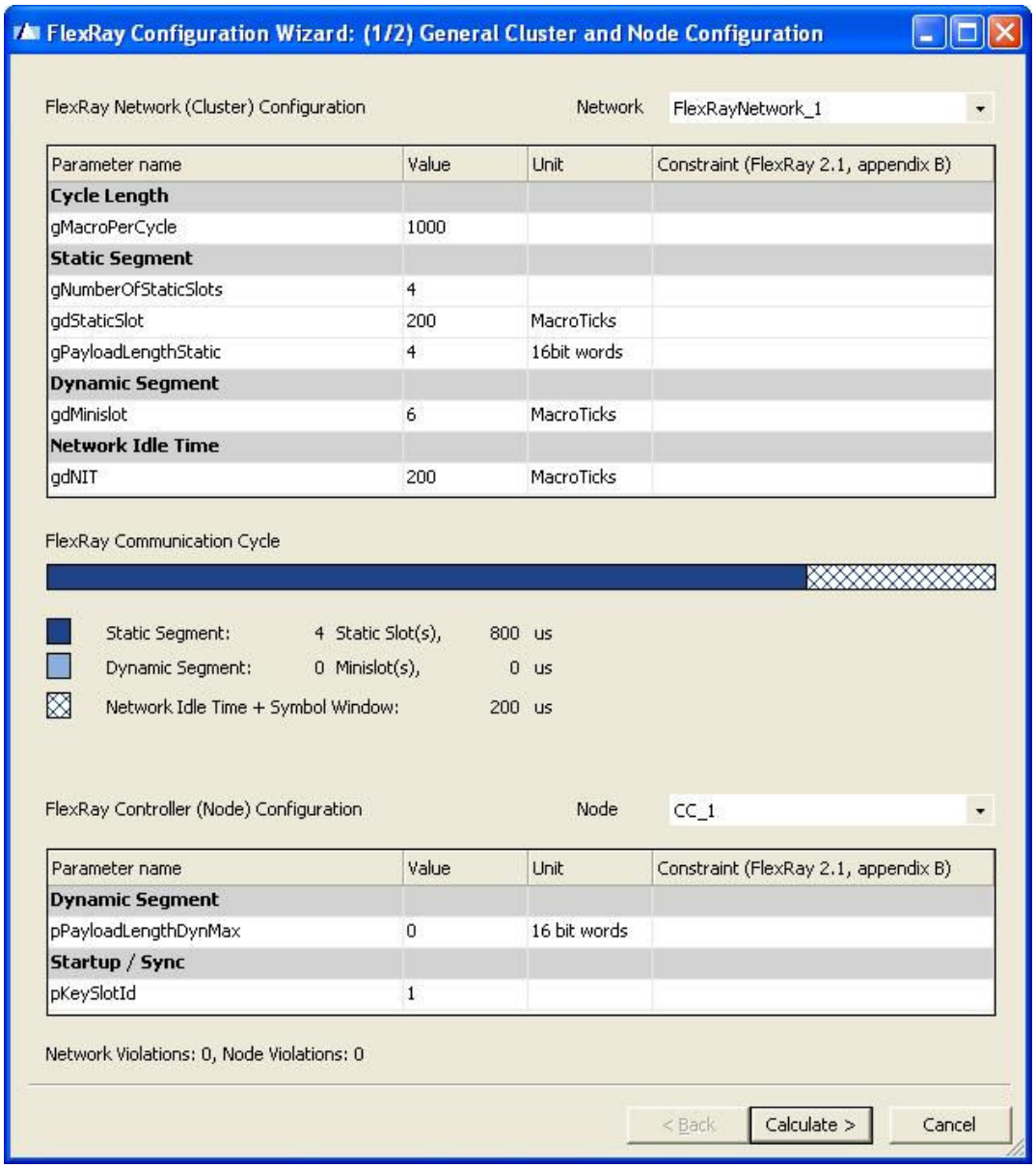

Abbildung 44 - FlexRay Configuration Wizard

Die Einheit Macroticks stellt die gemeinsame Zeitbasis des Netzwerks dar.
Aus der fest vorgegebenen Busgeschwindigkeit von 10Mbit/s und den Default-Werten zweier weiterer Parameter ergibt sich die Größe von einer Mikrosekunde für einen Macrotick.
Im unteren Teil des dargestellten Fensters folgen noch zwei weitere Parameter, die nicht für den gesamten Cluster gelten, sondern explizit für jeden einzelnen Knoten angegeben werden müssen:

- Maximale Payload-Länge für Dynamische Frames
- Key-Slot-ID zum Senden von Startup- und Sync-Frames

Sobald die getätigten Eingaben gegen das FlexRay-Protokoll verstoßen oder Unstimmigkeiten entstehen, wird dies durch eine Fehlermeldung angezeigt. Wenn alle Felder korrekt definiert worden sind, kann das Fenster über “Calculate“ verlassen werden.
Im darauffolgenden Fenster des *FlexRay Configuration Editors* sind sämtliche konfigurierbaren Parameter aufgelistet (vgl. Abbildung 45) und es können gegebenenfalls die beschriebenen Feineinstellungen vorgenommen werden. Für die vollständige Beschreibung dieser Parameter wird an dieser Stelle auf die FlexRay Protocol Specification [Flex2004] verwiesen.
Mit “Finish“ kann dieser zweite Arbeitsschritt abgeschlossen werden.

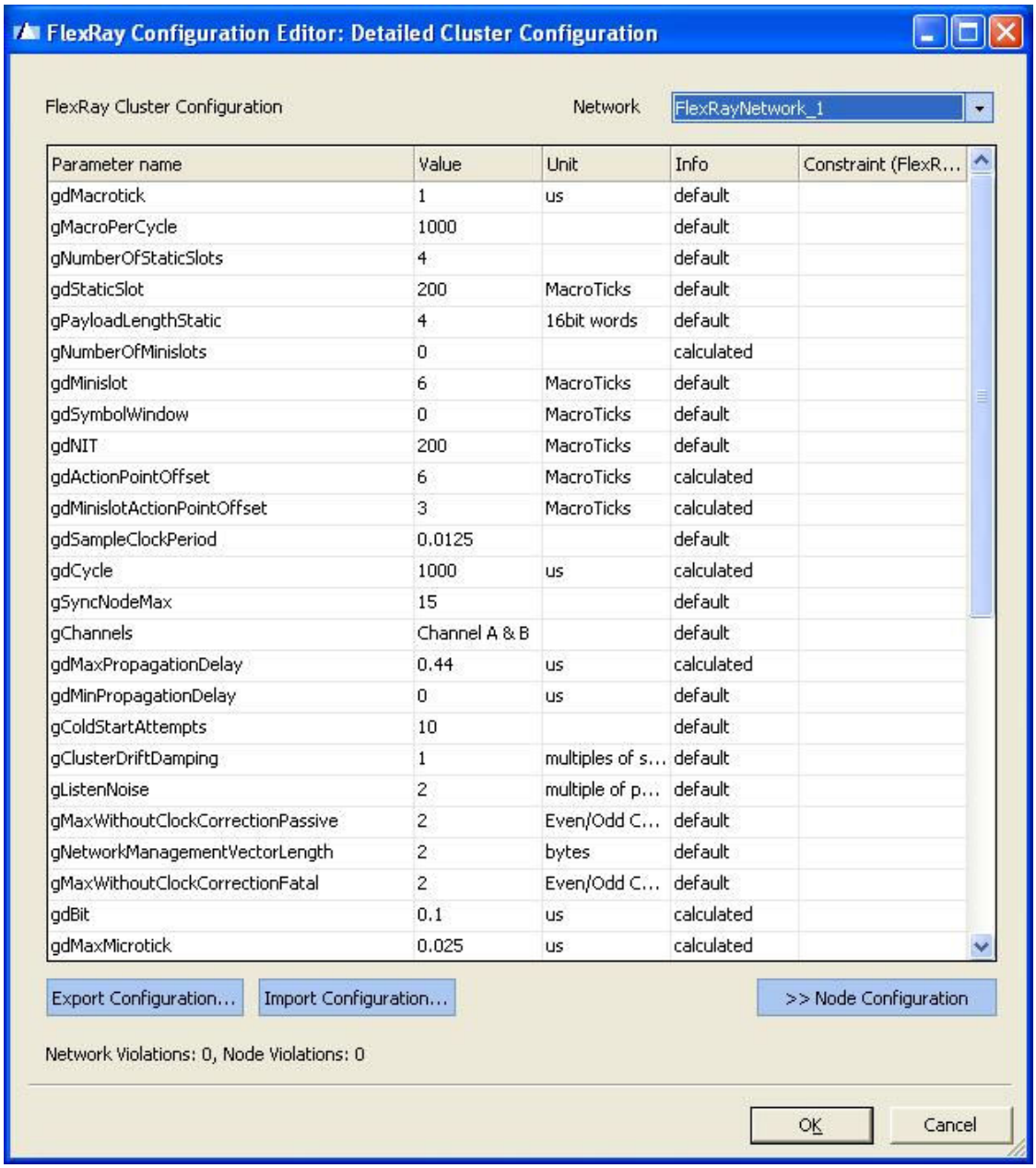

Abbildung 45 - FlexRay Configuration Editor

A.3.1.4 Kommunikationsplanung

Mit dem dritten Arbeitsschritt wird die Planung der Kommunikation auf dem FlexRay-Bus durchgeführt. Hierbei werden Sender, Empfänger und der Zeitpunkt der Übertragung festgelegt.

Frame-Signal-Editor
Mit dem ersten Menüpunkt *Frame-Signal-Editor* werden zunächst nur die Anzahl und die Größe der zu versendenden Frames definiert (vgl. Abbildung 46).
Die maximale Größe der Frames wurde bereits im vorigen Arbeitsschritt durch die Angabe der Payload-Länge konfiguriert. Es besteht nun die Möglichkeit bei bestimmten Frames ein Teil dieser Payload-Länge ungenutzt zu lassen und die default-mäßig festgelegten Namen zu ändern. Nachdem alle gewünschten Frames mit “Create“ erstellt worden sind, kann das Fenster mit “Close“ verlassen werden.

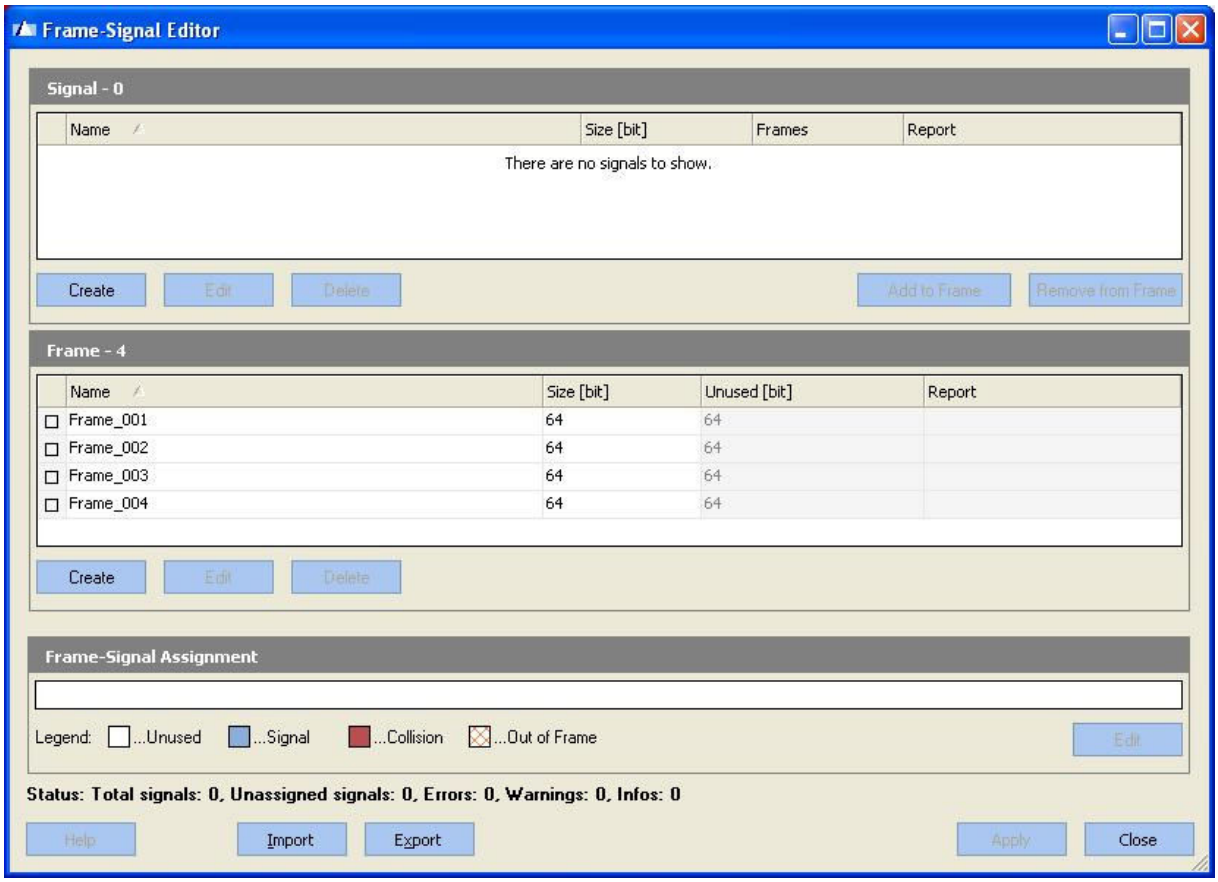

Abbildung 46 - Frame-Signal-Editor

Frame-Scheduler

Im Eintrag *Frame-Scheduler* werden die sogenannten *Frame-Triggerings* für das Statische und das Dynamische Segment erstellt (vgl. Abbildung 47).
Ein *Frame-Triggering* ist ein Container, der einen der zuvor erstellten Frames enthält und dem außerdem die folgenden Informationen zugewiesen werden müssen:

- Segment → Frame im Statischen oder Dynamischen Segment senden
- Slot → Nummer/ID des Slots, in dem gesendet werden soll
- CH → FlexRay-Kanal, auf dem gesendet werden soll
- BC → Basis-Zyklus, in dem erstmals gesendet werden soll
- CR → Zyklus-Wiederholung (z.B. = 1 : Frame in jedem Zyklus senden)
- TX Type → Sende-Modus (zyklisch, one-shot, ..)
- TX CC → Sender des Frames, Name des Kommunikationscontrollers
- RX CC → Empfänger des Frames, Name des Kommunikationscontrollers

Die im *Frame-Signal-Editor* erstellten Frames können auch mehreren *Frame-Triggerings* zugewiesen werden. Zum Beispiel könnte aus Redundanz-Gründen ein *Frame-Triggering* für das Senden auf Kanal A erstellt werden und ein zweites *Frame-Triggering*, bei dem der gleiche Frame im gleichen Slot auf Kanal B gesendet wird.
Die definierten Namen der *Frame-Triggerings* werden bei der späteren Implementierung der Anwendungssoftware verwendet, um dem jeweiligen Kommunikationscontroller die zu sendenden Nutzdaten zur Verfügung zu stellen.

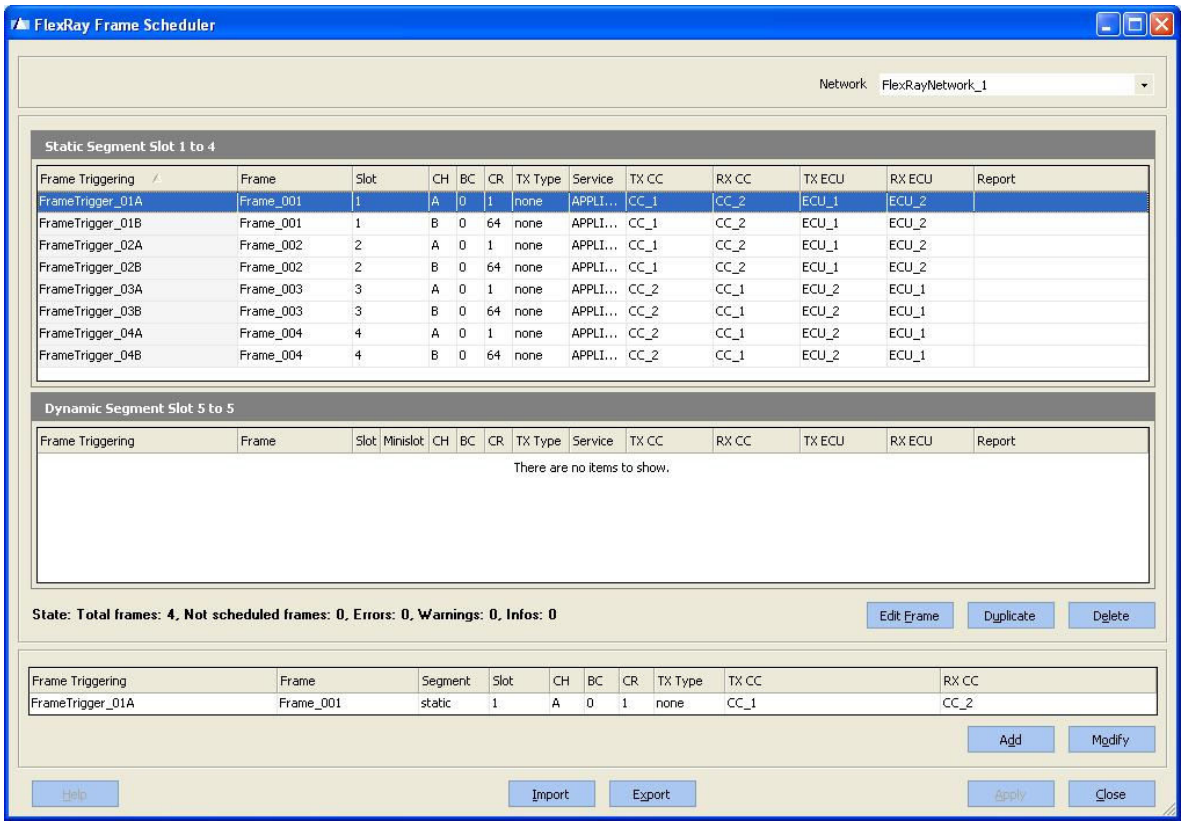

Abbildung 47 - Frame-Scheduler

A.3.1.5 ECU-Software

Im vierten Arbeitsschritt werden die später zu implementierenden Applikations-Tasks namentlich bekannt gemacht und es wird festgelegt, auf welchem Knoten sie ausgeführt werden. Ein Applikations-Task kann zum einen für die Bereitstellung der zu sendenden beziehungsweise zu empfangenden Daten definiert werden, und zum anderen für zusätzliche, zeitgesteuerte Aktivitäten des jeweiligen Knotens.

Applications-Task

Der erste Menüpunkt *Applications-Task* bietet die Möglichkeit durch “Create“ neue Tasks zu erstellen (vgl. Abbildung 48). Diese werden anschließend im generierten C-Code in einer Dispatcher-Tabelle zusammengefasst und, den zusätzlich anzugebenden Parametern entsprechend, zur Ausführung gebracht. Diese Parameter sind *Offset* und *Periode*, über die die Aktivierungszeitpunkte im Applikationszyklus festgelegt werden können, wobei über den *Offset* der erste Aktivierungszeitpunkt und über die *Periode* das Ausführungsintervall festgelegt wird. Des Weiteren sollte beim Entwurf ein Task vorgesehen werden, der typischerweise zum Ende des Applikationszyklus aktiviert wird und für die Synchronisation der lokalen Uhr eingesetzt wird.

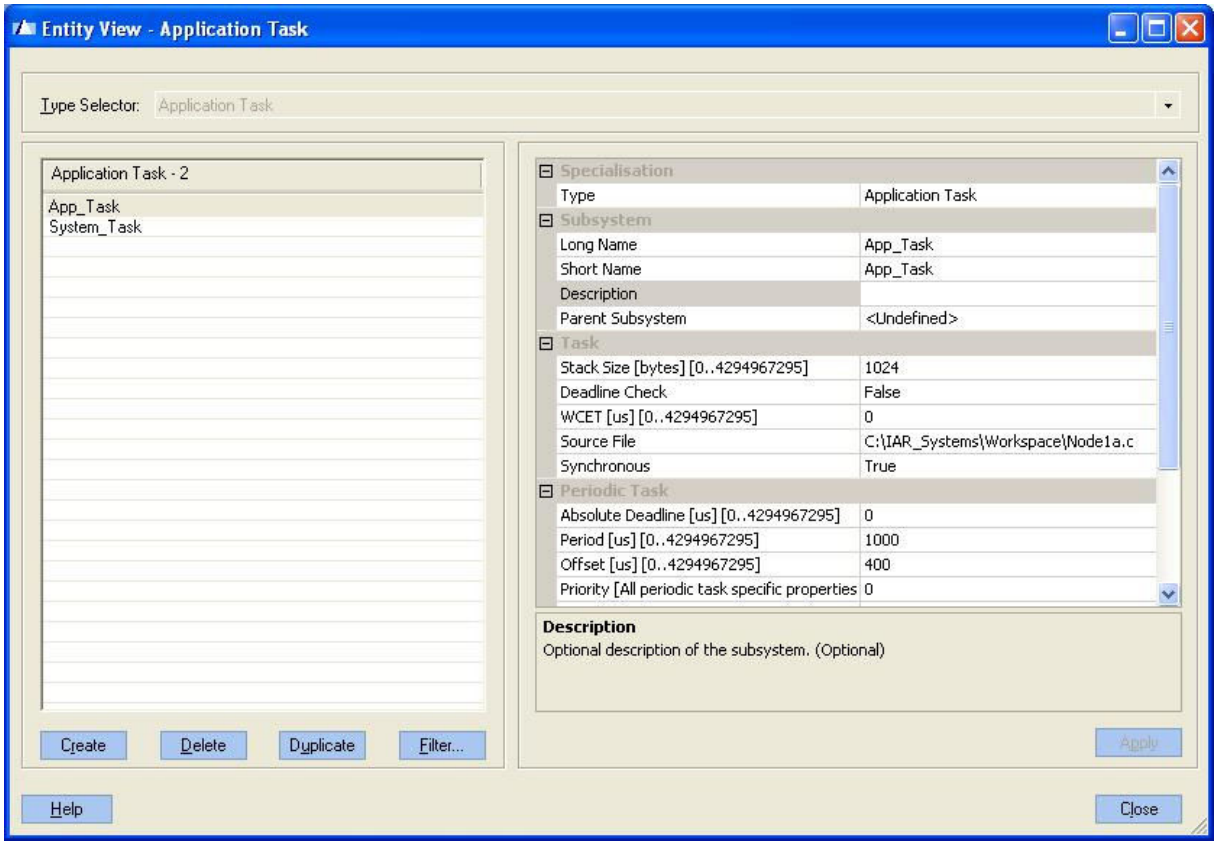

Abbildung 48 – Application Task

Subsystem to MCU/ECU

Nachdem alle benötigten Tasks erstellt worden sind, kann mit Hilfe des zweiten Menüpunktes *Subsystem to MCU/ECU* jedem Task ein bestimmter Knoten und die zugehörige MCU zugewiesen werden (vgl. Abbildung 49). Auch hier ist es möglich, die zuvor erstellten Tasks mehrfach zu verwenden. Zum Beispiel kann ein Task, der auf einem Knoten für das Senden eines Frames eingesetzt wird, auf einem anderen Knoten für das Empfangen dieses Frames zuständig sein. Wenn alle Tasks den entsprechenden Knoten zugewiesen sind, kann dieser Arbeitsschritt ebenfalls beendet werden.

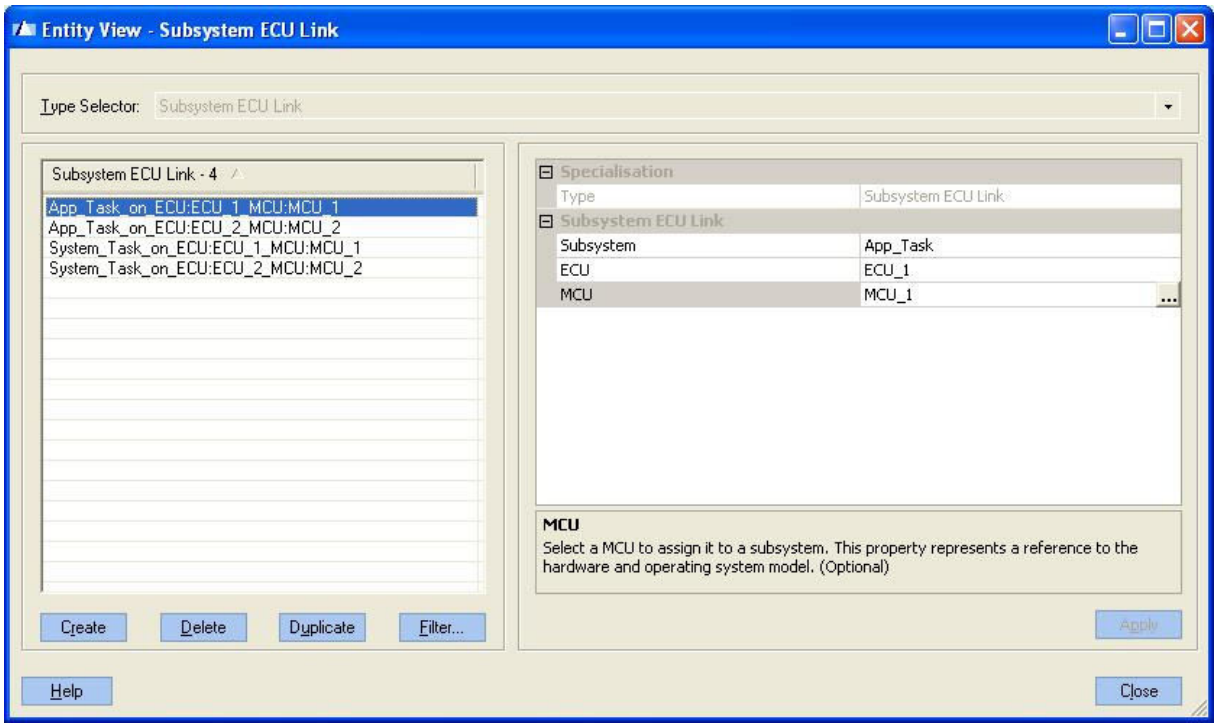

Abbildung 49 – Subsystem to MCU/ECU

A.3.1.6 ECU-Konfiguration

Im letzten Arbeitsschritt werden für jeden Knoten die Sende- und Empfangs-Puffer des Kommunikationscontrollers konfiguriert. Des Weiteren kann das für die Implementierung der Knoten verwendete Betriebssystem (z.B. AES) angegeben werden, das anhand der Dispatcher-Tabelle die zeitgesteuerte Ausführung der Tasks durchführt und die lokale Uhrzeit der jeweiligen MCU periodisch mit der globalen Zeit des FlexRay-Clusters synchronisiert.

ECU Driver Configuration

Nach der Auswahl dieses Menüpunktes öffnet sich ein Fenster, wie es in Abbildung 50 dargestellt ist. Im oberen Teil dieses Fensters (*Communication Assignment*) sind die den Knoten zugewiesenen Frame-Triggerings aufgelistet. Damit diese Frames auch vom jeweiligen Kommunikationskontroller gesendet bzw. empfangen werden können, muss jeweils ein Puffer für die Daten festgelegt werden, über den der Austausch der Daten mit dem Mikrocontroller ermöglicht wird. Dies geschieht im unteren Teil des Fensters (*Advanced Configuration*) durch Auswahl der Commstack-Registerkarte und dem darunterliegenden Menüpunkt *AutoBA* (*Automatic Buffer Assignment*). Durch Drücken der “Generate“-Taste werden den Frame-Triggerings nun automatisch Puffer zugewiesen, wobei die Zuweisung für jeden Knoten einzeln durchgeführt werden muss. Die Auswahl des jeweiligen Knotens kann am oberen Rand des Fensters getroffen werden (*Selector*).

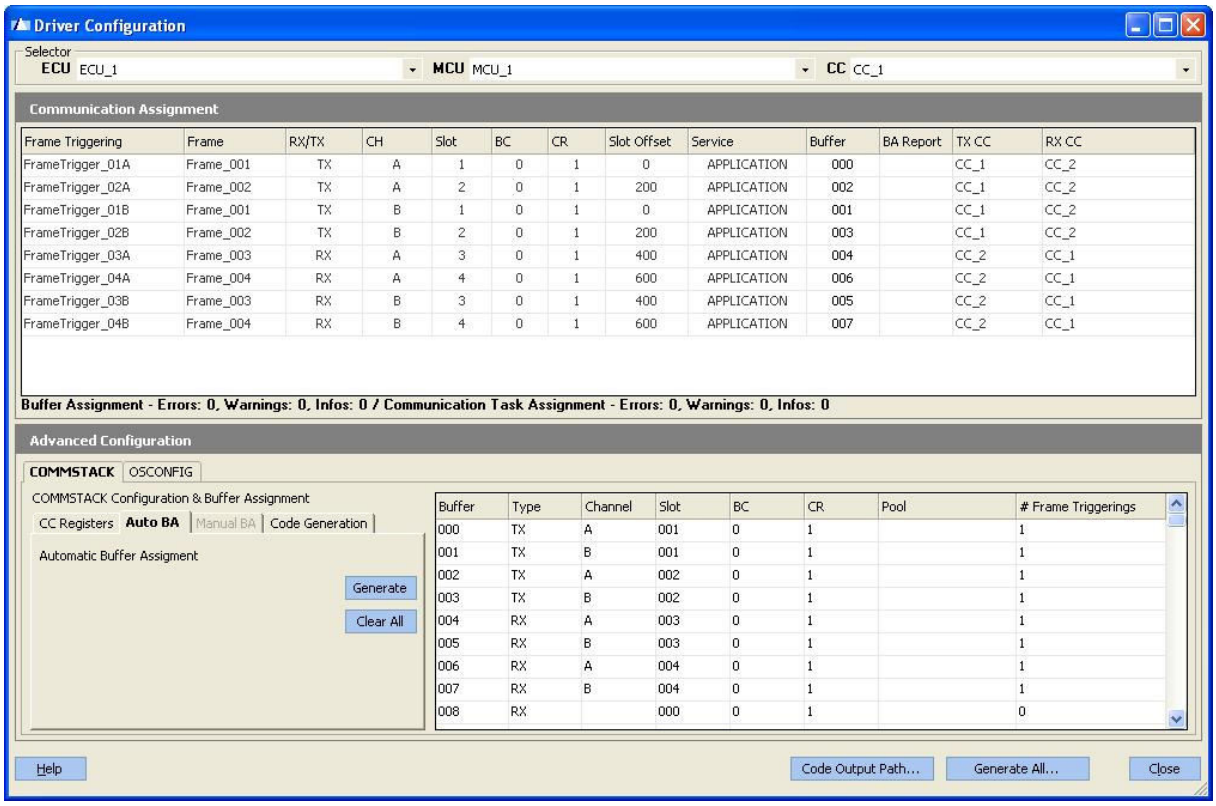

Abbildung 50 - ECU Driver Configuration, Commstack

Danach kann zur Register-Karte *OSCONFIG* gewechselt werden (vgl. Abbildung 51).
Über “Edit MCU“ öffnet sich ein weiteres Fenster, in dem die folgenden Konfigurations-Einstellungen(vgl. Abbildung 52) für die Mikrocontroller vorzunehmen sind:

- CPU Clock-Frequence → maximal 24 MHz
- OS Type → Decomsys AES
- Sync Type → Hard
- Max. Decrease → 50

Für den Fall, dass die lokale Uhr eines Knotens schneller als die globale Uhr des FlexRay-Systems läuft, ist Max Decrease der maximale Wert, der zur Korrektur eingesetzt wird.
Wenn alle Einstellungen vorgenommen worden sind, kann das Fenster über “Close“ geschlossen werden. Jetzt ist auch der letzte Arbeitsschritt mit dem Designer-Pro-Werkzeug abgeschlossen.
Am unteren Rand des *Driver Configuration*-Fensters muss nur noch über den Button “Code Output Path...“ ein Verzeichnis angegeben werden, in dem der vom Werkzeug generierte C-Code gespeichert werden soll. Nach dem Drücken von “Generate All..“ ist der C-Code erstellt und das DECOMSYS::DESIGNER Werkzeug kann beendet werden.

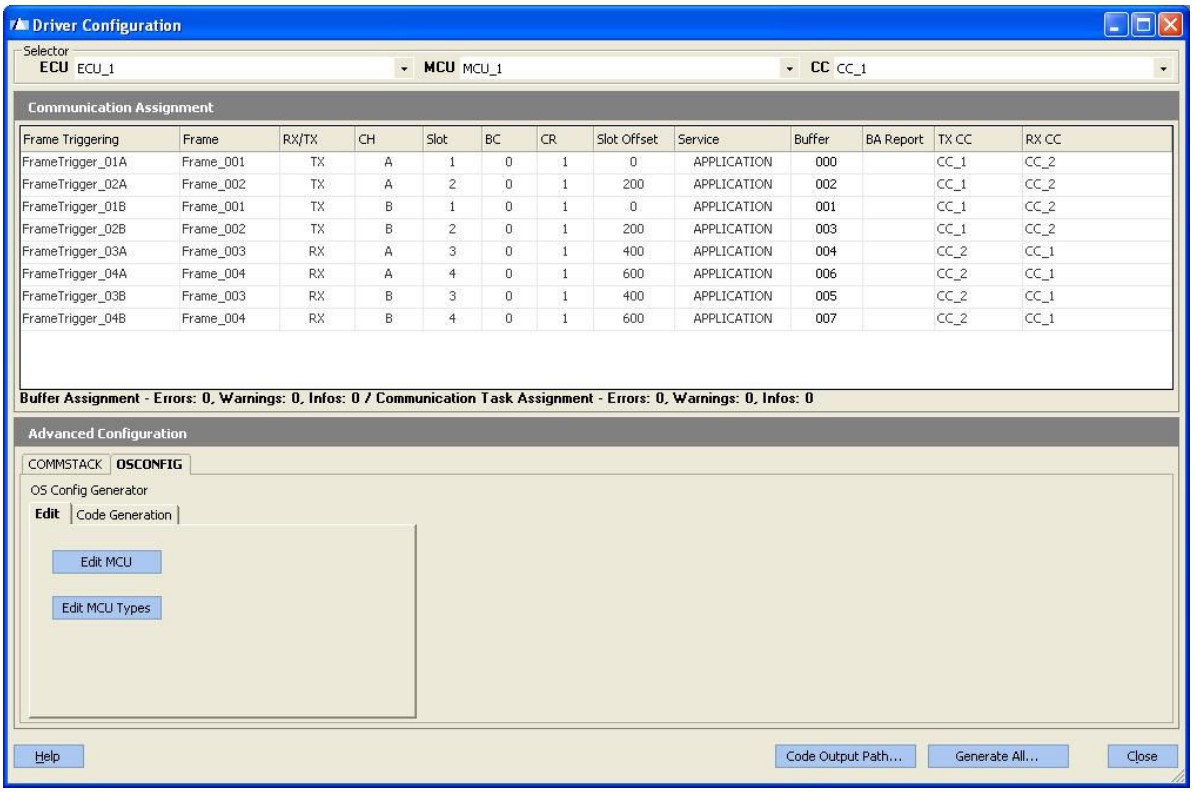

Abbildung 51 - ECU Driver Configuration, OSConfig

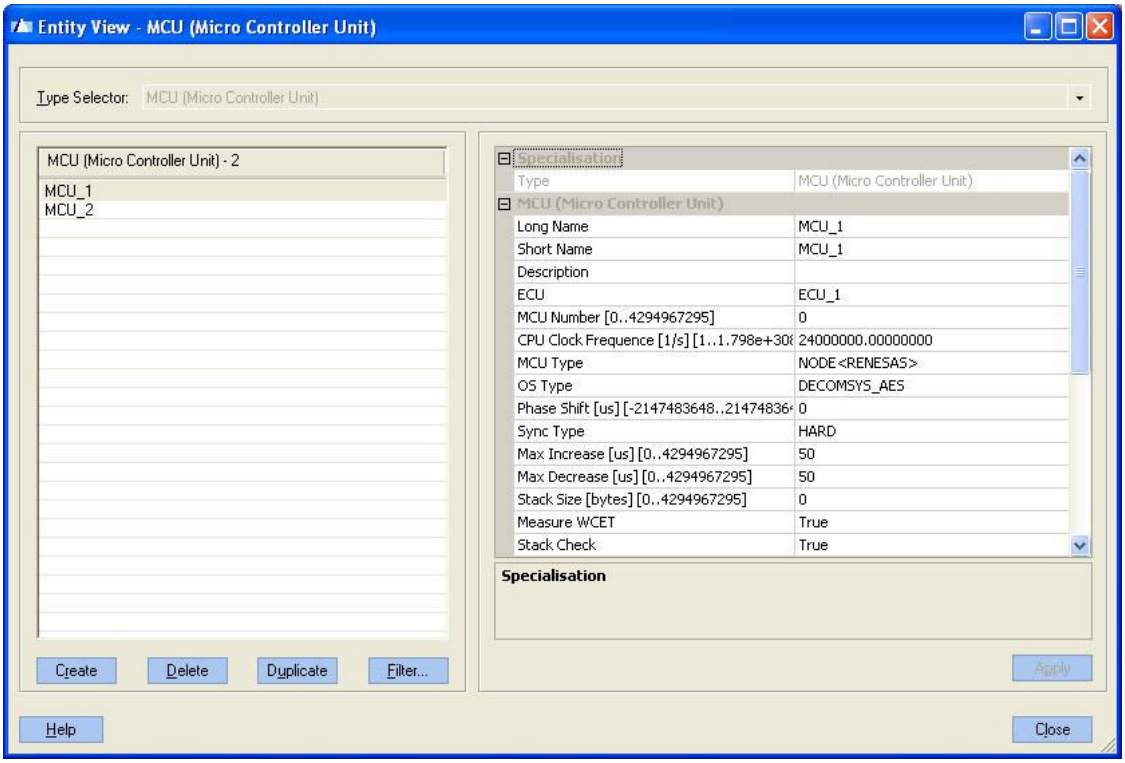

Abbildung 52 – MCU

A.3.2 IAR Embedded Workbench for Renesas M32C V3

Im folgenden Kapitel wird ein Entwurfsmuster dargestellt, das eine Möglichkeit zur Strukturierung der zahlreichen Dateien bietet. Des Weiteren wird beschrieben, wie der generierte C-Code des DECOMSYS::DESIGNER Werkzeugs eingebunden und verwendet wird. Des Weiteren werden die nötigen Einstellungen für die Node<Renesas>-Hardware, sowie für Compiler und Linker vorgegeben. Die zusätzlich benötigten Header-Dateien befinden sich im Installationsverzeichnis des StarterKits und die Build-Datei lnkM30855FW.xcl im Installationsverzeichnis der IAR Embedded Workbench.
Im Anschluss an die im Folgenden beschriebenen Arbeitsschritte ist nur noch die anwendungsspezifische Applikationssoftware zu implementieren (vgl. Abbildung 53).

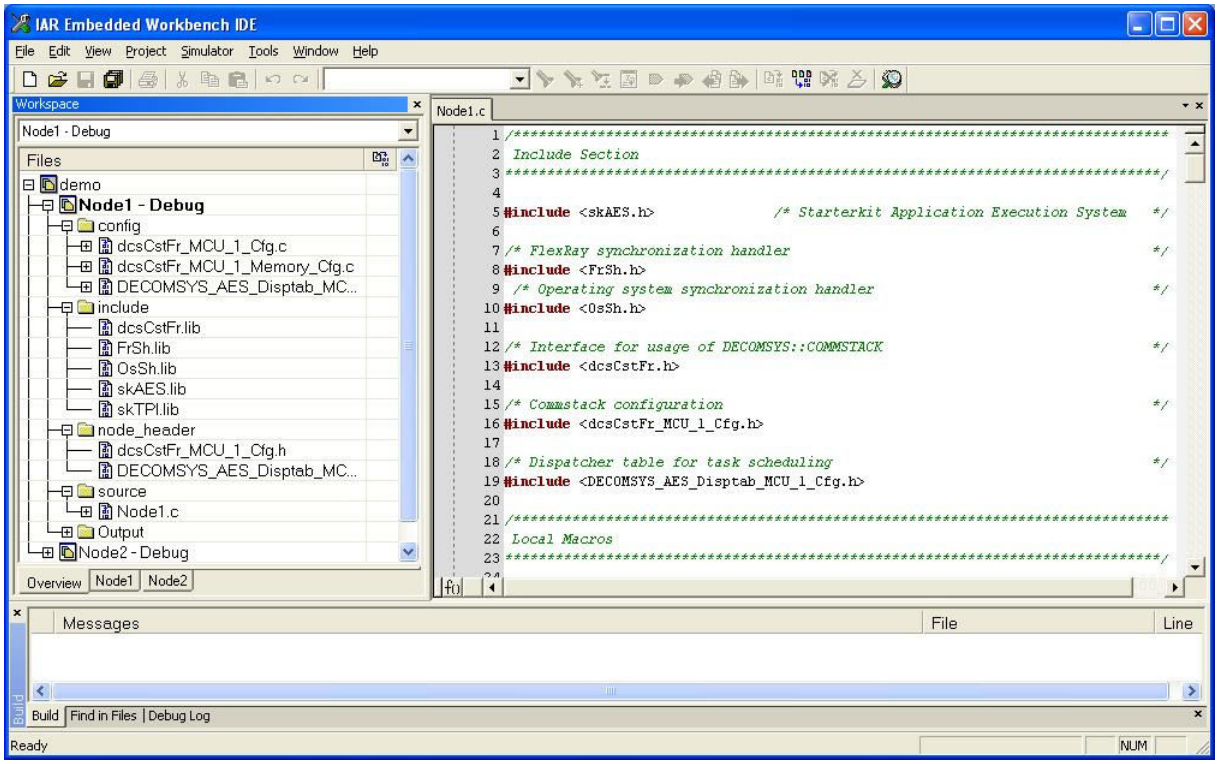

Abbildung 53 – IAR Embedded Workbench, Datei-Struktur

A.3.2.1 Datei-Struktur erstellen

Mit einem Datei-Manager ist zunächst die in Abbildung 54 dargestellte Datei-Struktur zu erstellen. Das übergeordnete Verzeichnis wäre typischerweise ein Ordner im Installationsverzeichnis der IAR-Workbench. Die überlappend dargestellte Verzeichnisstruktur von “node1_dir“ zeigt an, dass dieser Teil für jeden Knoten des geplanten Netzwerks zu erstellen ist.

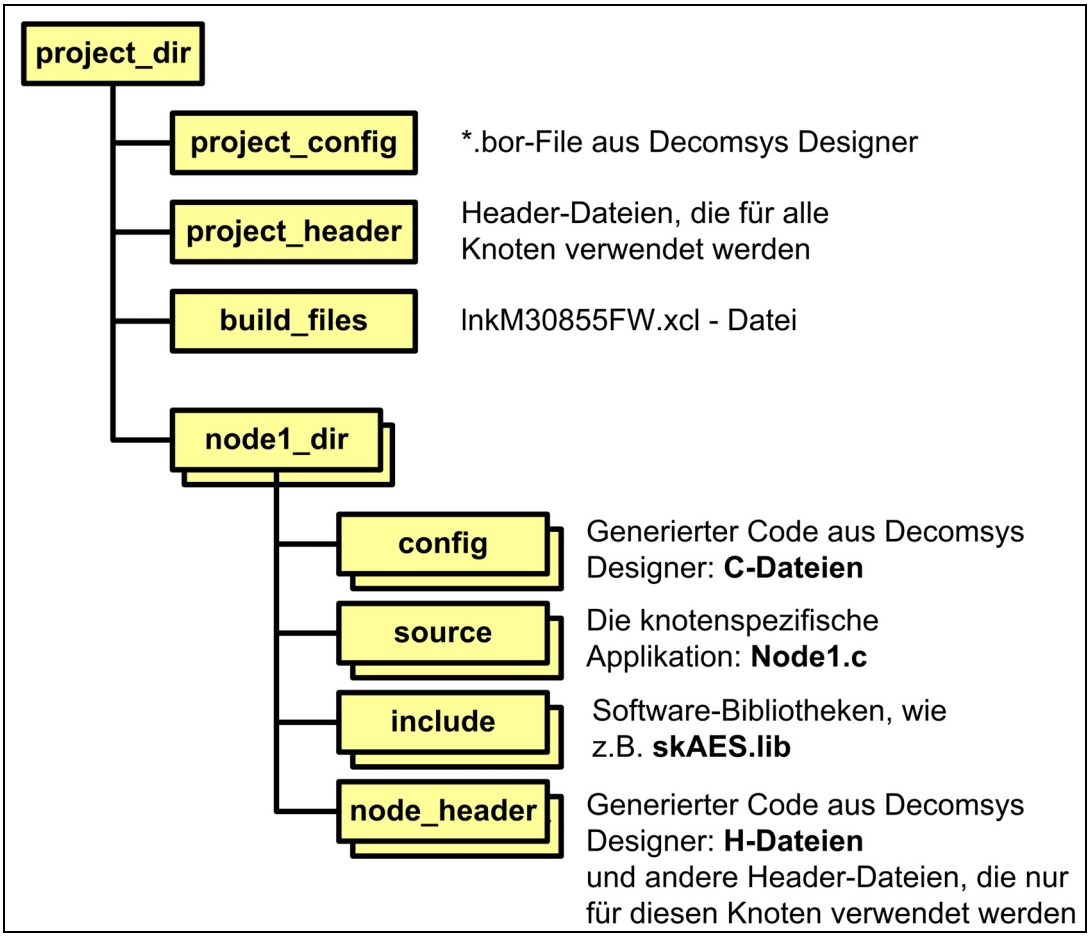

Abbildung 54 – Verzeichnis-Struktur

A.3.2.2 Projekte erstellen

Im Hauptmenü der IAR-Workbench wird für jeden Knoten ein eigenes Projekt im zuvor erstellten Verzeichnis angelegt. Zum Beispiel ist das Projekt für den Knoten 1 im Verzeichnis “project_dir\node1_dir\“ zu speichern:

Project → Create New Project → M32C (Empty Project)
→ “project_dir\node1_dir\node1.ewp”

A.3.2.3 Workspace erstellen

Mit dem folgenden Befehl werden die Projekte der einzelnen Knoten in einem Workspace zusammengefasst:

File → Save Workspace → M32C → “project_dir\<name>.eww”

A.3.2.4 Datei-Struktur in IAR übernehmen

Die mit dem Datei-Manager erstellte Struktur kann jetzt auch innerhalb der IAR so übernommen werden:

- Project → Add Group… ((Unter-)Ordner erstellen)
- Project → Add File… (die entsprechenden Dateien hinzufügen)

A.3.2.5 Implementierung der Applikationen

Anschließend können die knotenspezifischen Applikationen implementiert werden (z.B. Node1.c, …). Neben den Funktionen der Tasks, dessen Funktionsnamen im DECOMSYS:: DESIGNER definiert worden sind, müssen noch drei weitere Funktionen implementiert werden, die vom Decomsys-AES referenziert werden.

- void skAES_ApplInitHook(void)
- void skAES_ApplIdleTask(void)
- void skAES_ApplShutdownHook(skAES_ErrorType skAES_ErrNo)

Anstelle der main()-Funktion, die normalerweise den ersten Eintrag in einem C-Programm darstellt, wird dem Entwickler vom AES die Funktion skAES_ApplInitHook für die Initialisierung des anwendungsspezifischen Codes bereitgestellt. Für weitere Implementierungsdetails wird an dieser Stelle auf das AES- und Commstack-User Manual verwiesen [De2_2006] [De3_2006].

A.3.2.6 Compiler/Linker-Einstellungen

Bevor nun der Build-Vorgang für die Knoten gestartet werden kann, müssen die folgenden Compiler und Linker Einstellungen vorgenommen werden:

- Project → Options →
  - General Options → Target → Processor family = M32C/85
  - General Options → Target → Derivative = M30852FJ
  - General Options → Output → Output Directory => SetupDir-1 einfügen
  - General Options → Stack/Heap → Alles auf 1000 stellen
  - C-Compiler General → Optimizations → Medium
  - C-Compiler General → Preprocessor → Add includes => SetupDir-2 einfügen
  - Linker → Output → Output-File → Name in Node<#>.x30 ändern
  - Linker → Output → Format → Other → Output format: ieee-695
  - Linker → Output → Format → Other → Format variant: Renesas Compatible
  - Linker → Config → Linker Command File → SetupDir-3 einfügen

SetupDir-1 → General Options – Output – Output directories
Für Debug: $PROJ_DIR$\debug\obj\
Für Release: $PROJ_DIR$\release\obj\

SetupDir-2 → C-Compiler - Preprocessor-Anweisungen
$PROJ_DIR$\node_header\
$PROJ_DIR$\..\project_header\Types\
$PROJ_DIR$\..\project_header\skTPI\
$PROJ_DIR$\..\project_header\skAES\
$PROJ_DIR$\..\project_header\OsSh\
$PROJ_DIR$\..\project_header\FrSh\
$PROJ_DIR$\..\project_header\dcsCstFr\

SetupDir-3 → Linker – Config
$PROJ_DIR$\..\build_files\lnkM30855FW.xcl

A.3.3 KD3083-Debugger

Nach dem Build-Vorgang liegen die Binärdateien des Projektes vor und die Software kann mit dem Debugger KD3083 auf den entsprechenden NODE<RENESAS> übertragen werden.

A.3.3.1 Initialisierung

Nach dem Start des Debuggers KD3083 erscheint ein Konfigurations-Dialog, in dem folgende Werte einzustellen sind (vgl. Abbildung 55):

- Der FoUSB-Debugger ist via USB angeschlossen.
- MCU → M30855FW.MCU
- Compiler → IAR EWM32C
- Object Format → IEEE-695
- Processor Mode → Memory Expansion Mode

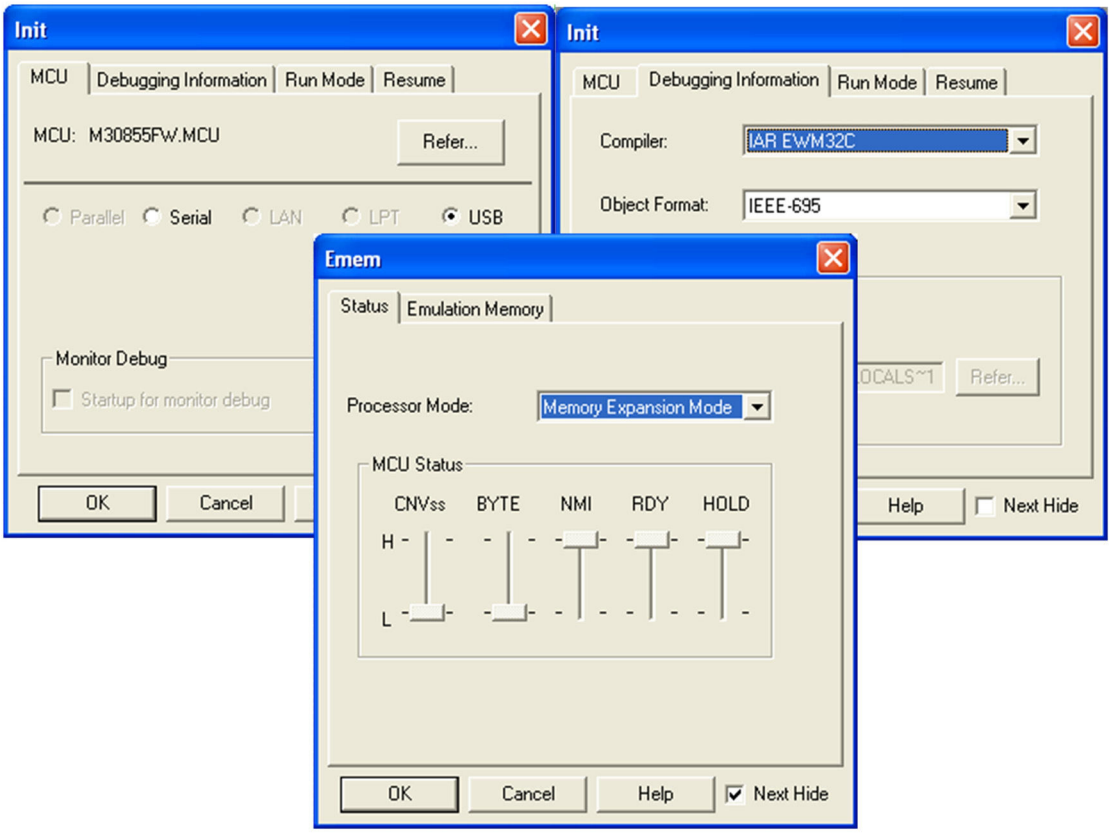

Abbildung 55 – Konfiguration des KD3083

A.3.3.2 Download & Debug

Nach den Initialisierungs-Einstellungen öffnet sich das Hauptfenster des Debuggers. Zum Herunterladen des zuvor kompilierten Programms, muss in der Menüleiste “File->Download->Load Module…” ausgewählt werden. Wenn das Entwurfsmuster und die Verzeichnisstruktur aus Kapitel A.3.2 verwendet worden ist, liegen die Build-Module der Knoten jeweils im Unterverzeichnis “obj“ der Node-Verzeichnisse (z.B. node1_dir\obj\node1.x30).

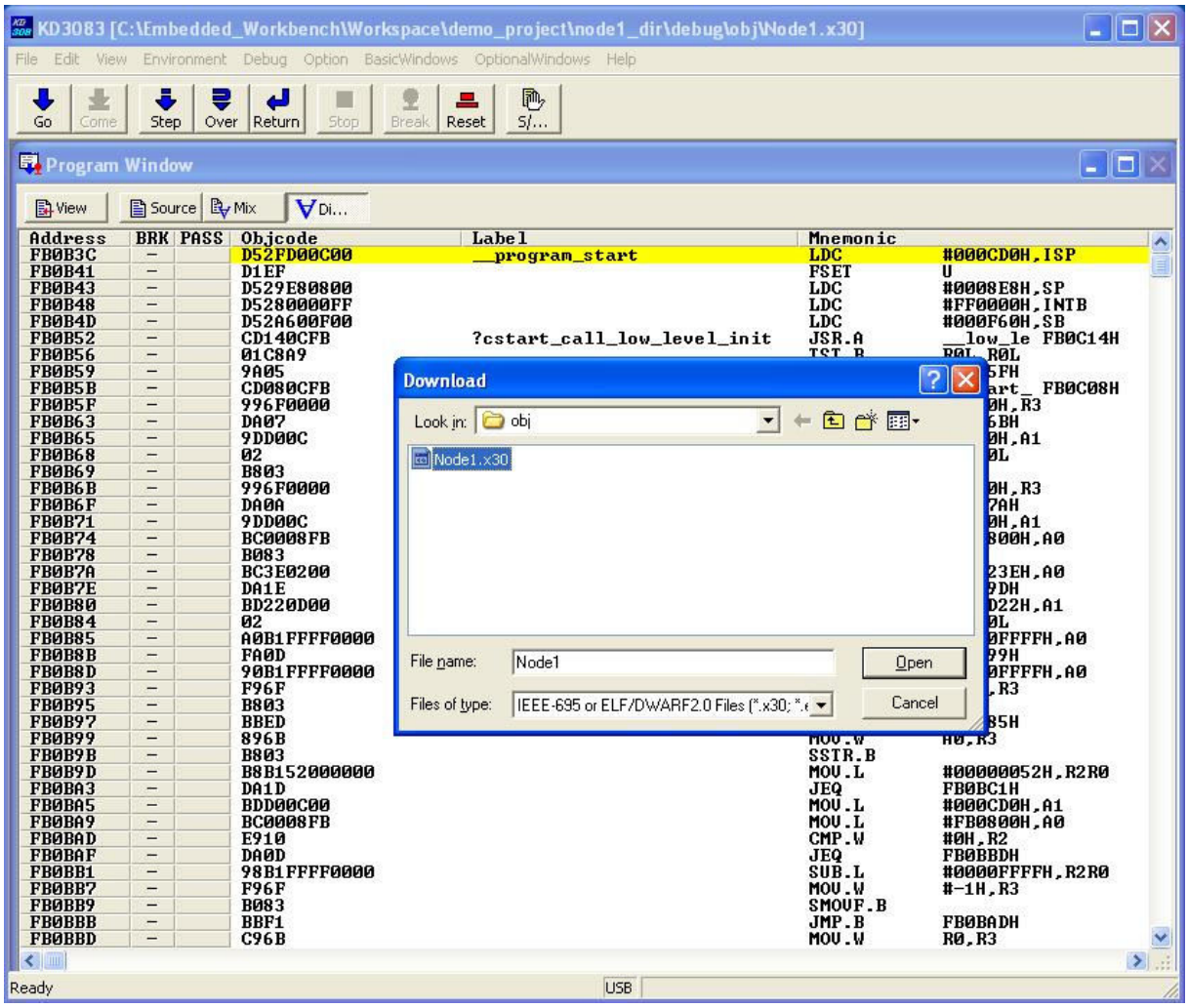

Abbildung 56 – Download des Programm-Codes im KD3083

Nach erfolgreichem Download kann durch das Drücken des "Go"-Buttons in der Symbolleiste des Hauptfensters das Programm auf dem Knoten gestartet werden. Um das Programm aus dem Flash-Speicher zu starten, muss der Debugger vom Zielsystem getrennt und ein Reset des Knotens durchgeführt werden. Im Menü-Eintrag “View“ kann für das Debuggen auch in die Quellcode-Ansicht gewechselt werden.

Versicherung über die Selbstständigkeit

Hiermit versichere ich, dass ich die vorliegende Arbeit im Sinne der Prüfungsordnung nach §22(4) ohne fremde Hilfe selbständig verfasst und nur die angegebenen Hilfsmittel benutzt habe.

Hamburg, 23.09.2008

Ort, Datum

Unterschrift