Markus Kogler, BSc

Aufsetzen einer Software Testumgebung für reale Steuergeräte

MASTERARBEIT

zur Erlangung des akademischen Grades

Diplom-Ingenieur

Masterstudium Elektrotechnik

eingereicht an der

Technischen Universität Graz

Betreuer

Dipl.-Ing. Dr.techn. Jürgen Fabian

Institut für Fahrzeugtechnik

Dipl.-Ing. (FH) Simon Waltenberger

AVL List GmbH

Gesperrt bis September 2021

Graz, September 2016

Danksagung

An dieser Stelle möchte ich mich bei all denjenigen bedanken, die mich während der Anfertigung dieser Masterarbeit unterstützt und motiviert haben.

Ganz besonders möchte ich Herrn Prof. Jürgen Fabian danken, der meine Arbeit betreut und begutachtet hat, für seine fachliche und persönliche Unterstützung sowie seine hilfreichen Anregungen und seine konstruktive Kritik.

Ein großer Dank gebührt meinem Kollegen Simon Waltenberger, der mir mit viel Geduld, Interesse und Hilfsbereitschaft zur Seite stand. Bedanken möchte ich mich für die zahlreichen interessanten Debatten und Ideen, die maßgeblich dazu beigetragen haben, dass diese Masterarbeit in dieser Form vorliegt. Danken möchte ich auch dem Unternehmen AVL List für die Ermöglichung dieser Arbeit und auch allen Kollegen, durch deren Anregungen und Unterstützung meine Masterarbeit kontinuierlich verbessert wurde.

Darüber hinaus möchte ich mich bei meinen Eltern Michaela Kogler und Walter Grinschgl bedanken, die mir mein Studium durch ihre Unterstützung ermöglicht haben und stets ein offenes Ohr für meine Sorgen hatten.

Ein abschließender Dank gilt meiner Freundin Michaela Lichtenegger, die mich immer wieder ermutigte und mich mit viel Geduld moralisch unterstützte.

Markus Kogler,

Graz, September 2016

EIDESSTATTLICHE ERKLÄRUNG

Ich erkläre an Eides statt, dass ich die vorliegende Arbeit selbstständig verfasst, andere als die angegebenen Quellen/Hilfsmittel nicht benutzt, und die den benutzten Quellen wörtlich und inhaltlich entnommenen Stellen als solche kenntlich gemacht habe. Das in TUGRAZonline hochgeladene Textdokument ist mit der vorliegenden Masterarbeit identisch.

Datum

Unterschrift

Kurzfassung

Durch die steigenden Anforderungen in der Industrie wird die Simulation von Gütern noch vor dem ersten Produktionsschritt immer wichtiger. Bei komplexen Systemen wie Kraftfahrzeugen werden Experten aus verschiedenen Disziplinen benötigt, so etwa die Bereiche Mechanik, Hydraulik und die in der Fahrzeugentwicklung immer wichtiger werdende Elektronik sowie Informationstechnologie. Für die Erstellung von Simulationsmodellen dieser Fachgebiete werden verschiedene Software-Tools verwendet. Um ein Zusammenspiel dieser Komponenten in einer Simulation zu ermöglichen, wird eine Co-Simulations-Software benötigt.

In der vorliegenden Masterarbeit wurde eine bestehenden Software Testumgebung, welche in MATLAB implementiert ist, modifiziert und damit den Test eines Steuergeräts mit einem Fahrzeugmodell ermöglicht. Um das simulierte Steuergerät gegen ein reales zu tauschen, wurde das von AVL List GmbH entwickelte Simulationsprogramm Model.CONNECT als Schnittstelle zwischen dem Echtzeitgerät und einem Rechner eingepflegt. Das Steuergerät kommuniziert in Echtzeit über CAN Schnittstellen und simulierten Sensoren/Aktoren mit dem am Computer laufenden Fahrzeugmodell.

Diese Vorgehensweise ermöglicht einen „Vorab-Hardware-in-the-Loop-Test“, welcher mit geringem Materialaufwand nicht zwingend am Prüfstand sondern auch in Büroräumlichkeiten durchgeführt werden kann. Dadurch können bereits in einem frühen Entwicklungsstadium die Funktionalitäten des Steuergeräts in Betrieb genommen und getestet werden.

Abstract

Due to rising requirements in industry the simulation of products preliminary production gains in significance. In developing complex items there is a need for experts from different technical fields, like the mechanical and hydraulic domain and furthermore the growing importance of electronics as well as information technology in automotive engineering. To build up simulation models there are different software environments for this specialised fields. To enable cooperation of these components, co-simulation is needed.

In the current master thesis an existing software test-environment implemented in MATLAB was modified to realize the test of a control unit with a plant model. To replace the virtual control unit with a real one, the AVL List Ltd. developed simulation program Model.CONNECT was used as an interface between the real-time device and the computer. The control unit communicates in real-time over CAN busses and simulated sensors/actuators with the computer-based plant model.

This approach allows a “Pre-Hardware-in-the-Loop-Test”, which can be not only performed in a laboratory, but also in office rooms. Thus it is possible to test functionalities in an early stage of development.

Abkürzungen

Abkürzung	Bedeutung
ABS	Antiblockiersystem
ADD	Automotive Data Dictionary
ASIL	Automotive Safety Integrity Level
BMS	Battery Management System
BSW	Basic Software
CAN	Controller Area Network
CD	Compact Disk
CIL	Custom Interface Layer
CO_2	Kohlendioxid
CRC	Cyclic Redundancy Check
DBC	Data Base CAN
DC	Direct Current
DRE	Discrete Response Equivalent
ECCO	Energy-Conservation-based Co-Simulation
ECU	Electronic Control Unit
EMV	Elektromagnetische Verträglichkeit
ESP	Elektronisches Stabilitätsprogramm
EV	Electric Vehicle

Abkürzung	Bedeutung
HCU	Hybrid Control Unit
HiL	Hardware in the Loop
HV	Hochspannung
HVAC	Heating, Ventilation and Air Conditioning
HVCU	Hybrid Vehicle Control Unit
HVIL	High Voltage Interlock
HW	Hardware
I/O	Input / Output
IBS	Intelligent Battery Sensor
ICOS	Independent Co-Simulation
ID	Identification
IP	Internet Protocol
KFZ	Kraftfahrzeug
LIN	Local Interconnect Network
MATLAB	Matrix Laboratory
MiL	Model in the Loop
NEPCE	Nearly Energy Preserving Coupling Element
NI	National Instruments
NTC	Negative Temperature Coefficient

Abkürzung	Bedeutung
PC	Personal Computer
PDIP	Plastic Dual In-line Package
RAM	Random-Access Memory
SiL	Software in the Loop
SOC	State of Charge
SUV	Sport Utility Vehicle
SW	Software
TCU	Transmission Control Unit
UDP	User Datagram Protocol
USB	Universal Serial Bus

Symbolverzeichnis

Symbol	Bedeutung
U …	Spannung
I …	Strom
R …	Ohmscher Widerstand
P …	Leistung
p …	Druck
V …	Verstärkungsfaktor
δT …	Mikrozeitschritt
ΔT …	Makrozeitschritt

Inhalt

1 Einleitung

Seit dem verstärkten Einzug der Elektronik im Fahrzeug zu Beginn der 1980er Jahre hat sich die Anzahl der Steuergeräte im Fahrzeug deutlich erhöht [1]. Es wurden immer mehr Funktionen für Sicherheit oder Komfort des Fahrers zur Verfügung gestellt, die heute einen Standard in der Automobilbranche darstellen, angefangen von der Steuerung und Überwachung des Antriebsstrangs, über sicherheitsrelevanten Aufgaben wie ABS oder ESP, bis hin zu Navigationssystemen und Vernetzung von Mobiltelefonen. Die steigende Anzahl an Steuergeräten führt neben einem erhöhten Verkabelungsaufwand mit einhergehender Kosten- sowie Gewichtssteigerung auch zu größeren Aufwendungen in der Qualitätssicherung [2].

Die Entwicklung von Prüfsystemen und Prüfkriterien von den immer komplexeren Gesamtsystemen stellt eine enorme Herausforderung dar. Da die Aktivität einzelner Systeme wenig Aussagekraft auf das Verhalten im Verbund hat, gewinnen holistische Simulationen von Gesamtsystemen [3] zunehmend an Bedeutung. In Oberklassefahrzeugen sind bereits um die 80 bis 100 Steuergeräte verbaut (2012, [2], [4]), welche größtenteils über CAN Bus miteinander kommunizieren und die als vernetzter Zusammenschluss nur mit extremen Aufwand getestet werden können. Eine Möglichkeit liegt in der Validierung mittels einer Co-Simulation, in der Modelle sowie Echtzeitgeräte mit Sensoren und Aktoren eingebunden werden können.

1.1 Motivation

Der Konnex von Modellen unterschiedlicher technischer Fachrichtung gibt bereits in frühen Stadien der Systementwicklung Einsicht in das Gesamtsystemverhalten. Werden in weiterer Folge reale Geräte eingebunden, steigen die Aussagekraft der Ergebnisse, aber auch die Anforderungen an die eingebunden Systeme und der Simulationswerkzeuge [5].

Die Entwicklung von Hybridfahrzeugen mit Verbrennungs- und Elektromotoren stellt eine weitere Herausforderung dar, da nun zusätzlich zahlreiche weitere elektrische und elektronische Komponenten benötigt werden. Neben weiteren Steuergeräten müssen auch Energiespeicherung und -umwandlung, wie auch Sicherheit bei hohen Spannungen und Fahrstrategien berücksichtigt werden.

Die Inbetriebnahme eines Steuergeräts findet im Prototypenfahrzeug statt und der Test jeder einzelnen Funktion kostet noch vor dem ersten gefahrenen Meter eine Menge an

Entwicklungszeit. Durch Einbindung des Steuergeräts in eine Echtzeit-Co-Simulation mit genauen Fahrzeugmodellen und Bereitstellung aller notwendigen Hardwareschnittstellen können bereits in den Büroräumlichkeiten viele Funktionalitäten und Systemeigenschaften getestet werden.

Bei dem Fahrzeug für das aktuelle Projekt handelte es sich um einen SUV, der auf ein Hybridfahrzeug in P2-Konfiguration umgebaut wurde. Der Verbrennungsmotor ist eine Eigenentwicklung von AVL List GmbH und arbeitet nach dem Miller-Verfahren. Mit Hilfe des Elektromotors wird vor allem im Stadtverkehr der Verbrauch und somit die CO_2 Bilanz verbessert. Rein elektrisches Fahren ist mit dem relativ schwachen Elektromotor und der geringen Batteriekapazität nur für sehr kurze Strecken möglich.

1.2 Aufgabenstellung

Die Aufgabenstellung dieser Masterarbeit war es, die bestehenden Tests der Software für ein Steuergerät, konkret für ein Hybridsteuergerät, zu erweitern, um einen aussagekräftigeren Funktionstest zu erhalten. Das Testsystem sollte die Kommunikation eines virtuellen Fahrzeugmodells mit einem realen Steuergerät ermöglichen und überwachen. Zu den Randbedingungen zählte die Weiterbenutzung der MATLAB basierten AVL List GmbH Entwicklungsumgebung AVLab, und der Einsatz der Co-Simulationsumgebung Model.CONNECT.

Aufgrund der Vielzahl an Steuergeräten wurde eine Struktur für den Test eines Hybridsteuergeräts mit einem Fahrzeugmodell eines parallelen Hybridfahrzeugs in P2 Konfiguration entwickelt. Als Grundlage für die Entwicklung der Testmethode lag eine Softwareversion vor, deren Funktion bereits erfolgreich im Fahrzeug getestet wurde und das Testsystem sollte bei erfolgreichem Aufbau auf den neuesten Software-Stand angepasst werden.

1.3 Ziele

Zur Validierung der Co-Simulation wurden die Simulationsergebnisse mit den Ergebnissen des ursprünglichen MiL-Tests validiert. Auch wenn eine exakte Übereinstimmung der Daten nicht möglich war, konnte zumindest anhand der Signalverläufe eine Aussage über das Systemverhalten getroffen werden und gegebenenfalls Simulationsparameter angepasst werden.

Weiterfolgend sollte es möglich sein, auch andere dem Antriebsstrang zugehörige Steuergeräte zu testen beziehungsweise mehrere Steuergeräte in einem Verbund mit Model.CONNECT zu vernetzen und einen Test des Gesamtsystems durchzuführen.

2 Computersimulation

Mit den steigenden Anforderungen an die Qualität und Lebensdauer von Systemen ist die Simulation zu einem wichtigen Werkzeug in der Produktentwicklung geworden. Es haben sich für viele technische Bereiche spezielle Simulationsprogramme entwickelt, welche auf die spezifischen Problemstellungen abgestimmt sind. Mit dem Fortschritt in der Computertechnologie können in kürzerer Zeit immer größere Systeme modelliert und simuliert werden.

Beginnend von der Abstraktion von Problemstellungen über die Modellbildung können in Entwicklungsumgebungen die Systeme als Differentialgleichungssysteme aufgebaut werden.

2.1 Solver

Zur numerischen Lösung dieser mathematischen Modelle werden Integrationsverfahren (Gleichungslöser, Solver) eingesetzt, die mit unterschiedlichen mathematischen Verfahren arbeiten [6].

Es gibt Solver für diskrete und kontinuierliche Modelle, mit fester und variabler Schrittweite, die, abhängig von ihrer Ordnung, eine bestimmte Anzahl der letzten vorangegangenen Ergebnisse für die Berechnung des nächsten Simulationsschritts heranziehen. Manche Integratoren eignen sich besser für sogenannte steife Systeme, welche langsame und schnelle Dynamiken haben [7], andere sind auf gute Stabilität oder hohe Genauigkeit getrimmt [6]. Weitere Gleichungslöser, wie DRE, zielen auf spezielle Einsatzgebiete wie die Echtzeitsimulation ab [7].

2.2 Co-Simulation

Für moderne Entwicklungsprozesse ist die Simulation von Gesamtsystemen zu einem wichtigen Bestandteil geworden. Durch die steigende Komplexität der verschiedenen technischen Disziplinen haben sich Modellierungsumgebungen für spezielle Problemstellungen entwickelt. Die Aufgabe einer Co-Simulationsumgebung liegt in der Kopplung der Simulationswerkzeuge und dem korrekten Austausch der Simulationsdaten. Die Simulation der einzelnen Modelle erfolgt in ihrer jeweiligen Simulationsumgebung. Aus der Sicht des Co-Simulations-Programms entsprechen die einzelnen Modelle einer Black Box, von denen nur die Ein- und Ausgänge bekannt sind [3], [8].

Zusätzlich zu den virtuellen Simulationsmodellen sollen auch Anknüpfungspunkte zu Echtzeitgeräten vorhanden sein, etwa die Vernetzung mittels eines Bussystems (siehe Kapitel 2.11.2).

2.3 Definition

Abhängig von der Anzahl der Solver und der Modellierungsumgebungen gibt es Versuche zur Begriffsvereinheitlichung von Simulationsanordnungen (Abbildung 2.1). Bei der klassischen Simulation löst ein Solver ein Modell (III). Um hier komplexe Modelle zu simulieren, muss eine interdisziplinäre Modellentwicklungsumgebung gefunden werden. Zusätzlich muss der Gleichungslöser in der Lage sein, den gesamten Anforderungen an Genauigkeit und Simulationszeit gerecht zu werden. Um die Berechnung eines Modells zu beschleunigen, ist es sinnvoll es in Subsysteme zu gliedern und parallel zu rechnen. Die Modellbildung ist weiterhin geschlossen mit einer Umgebung, jedoch wird die Simulation verteilt (IV). Mit dieser Vorgehensweise kann man steife Gleichungsteile abspalten und mit geeigneten Solvern lösen.

Bei der Modellkopplung werden einzelne Modellsubsysteme aus unterschiedlichen Entwicklungsumgebungen zusammengefasst und mit einem Solver simuliert (I). Die Modellteile könne dabei als Gleichungen oder Programmcode verknüpft oder als Funktion aufgerufen werden.

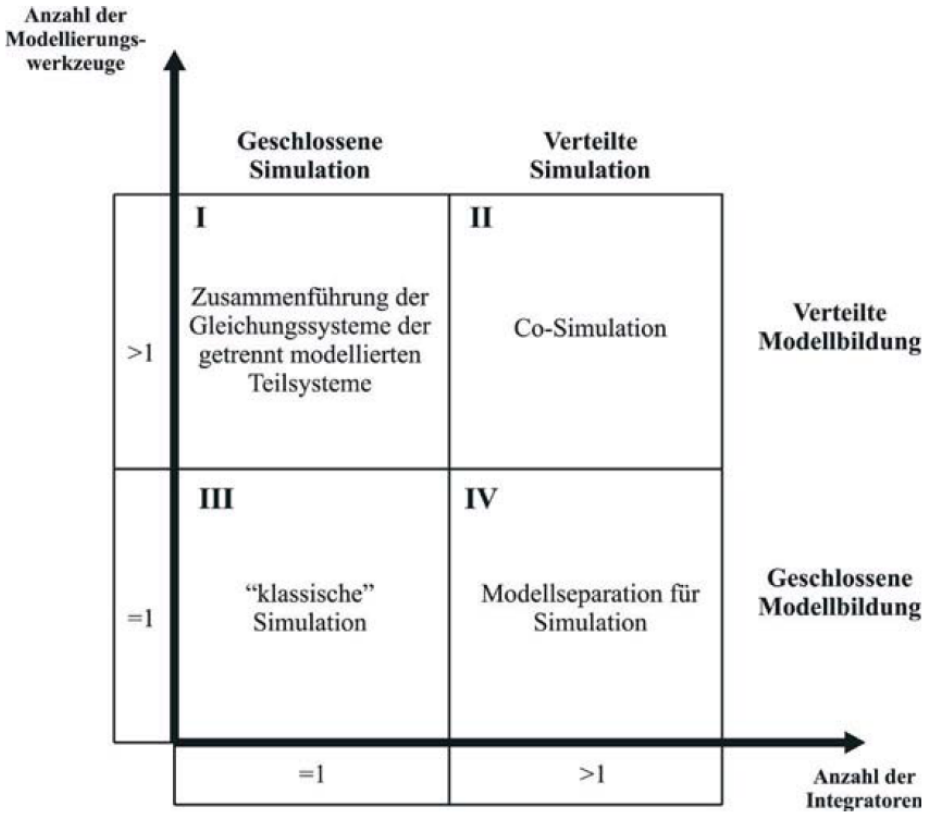

Abbildung 2.1: Simulationsvarianten [8]

Der Begriff Co-Simulation gilt nun bei der Verwendung von Subsystemen aus mehreren Modellierungsumgebungen (II), die mit unterschiedlichen Solvern gelöst werden [8], [9].

Es ist dabei ein Programm notwendig, welches Schnittstellen zu den Modellen besitzt und den Ablauf einer Co-Simulation verwaltet.

2.4 Herausforderungen

Die Aufgaben einer Co-Simulationsumgebung sind vielfältig und beschränken sich nicht nur auf die Bereitstellung von Schnittstellen zu verschiedenen Modellierungsumgebungen. Vielmehr werden über Algorithmen ein korrekter Datenaustausch gewährleistet und Schwierigkeiten bei der Kopplung von Modellen unterschiedlicher Dynamik beigekommen. Ein wichtiger Aspekt gilt auch der Überwachung der zwischen den Modellen ausgetauschten Gesamtenergie [5]. Werden die Simulationsmodelle für eine Co-Simulation adaptiert, sind gewisse Aspekte zu berücksichtigen. So dürfen die Modelle in ihrer Funktion nicht verändert, nur Ein- und Ausgänge hinzugefügt werden. Da die Modelle von Entwicklern verschiedener Fachbereiche erstellt wurden und es für die Simulationsparameter viel spezifisches

Wissen um das Gebiet und das Modell erfordert, dürfen auch die Schrittweiten und Solver nicht verändert werden [3].

2.5 Anwendungen

Bei der Entwicklung von Hybridfahrzeugen kann der Einsatz von Co-Simulation von großem Nutzen sein. Neben den mechanischen, thermischen und hydraulischen Komponenten bis dato klassischer Fahrzeuge, kommt ein komplexes elektrisches System zum Einsatz welches neben weiteren Steuergeräten auch um zahlreiche elektronische Komponenten erweitert wird [10]. Das Zusammenwirken der vielen einzelnen Module und Modelle kann durch die vernetzte Simulation aller Teile besser vorhergesagt werden.

2.5.1 Hybridfahrzeuge

Ein Hybridfahrzeug ist definiert als Fahrzeug mit mindestens zwei unterschiedlichen Energiequellen und mindestens zwei unterschiedlichen Antrieben [11]. In den meisten Fällen handelt es sich um eine Kombination aus Verbrennungsmotor und Elektromotor, es ist aber auch der Einsatz von Brennstoffzellen zur Erzeugung elektrischer Energie denkbar. Abhängig von der Anordnung der Antriebe ist eine Einteilung von Hybridfahrzeuge nach ihrer Grundstruktur möglich.

Mit Hilfe von Hybridfahrzeugen können die strenger werdenden Vorschriften für Abgasemissionen besser erfüllt werden und stellen das Bindeglied zwischen Fahrzeugen mit Verbrennungsmotor und Elektrofahrzeugen dar. Es gibt einen Kompromiss zwischen Reichweite und lokal anfallenden Abgasemissionen.

2.5.2 Serieller Hybridantrieb

Bei einem seriellen Hybridfahrzeug hat der Verbrennungsmotor keine mechanische Verbindung zu den antreibenden Rädern (Abbildung 2.2, oben). Er treibt einen, meist im generatorischen Betrieb arbeiteten Elektromotor an, welcher die Antriebsenergie für einen weiteren Elektromotor liefert. Als Energiespeicher befindet sich zwischen dem Generator und dem Traktionsmotor eine Batterie. Ist es möglich, diese Batterie über eine externe Ladestation zu laden, so spricht man auch von einem Plug-In Hybrid. Der antreibende Elektromotor muss das gesamte Antriebsmoment zur Verfügung stellen und dementsprechend groß ausgelegt werden.

Der Verbrennungsmotor wird immer im optimalen Lastpunkt betrieben, was zu einer Treibstoffeinsparung führt und kann unabhängig vom Fahrzustand die Batterie laden.

Der Nachteil liegt in der Wirkungsgradkette des Energieflusses, an der neben dem Verbrenner auch zwei Elektromotoren, die Batterie und ein Gleich- bzw. Umricher beteiligt sind. Durch die vielen beteiligten Module wird der Antrieb schwerer und es entsteht ein erhöhter Wartungsaufwand [11].

2.5.3 Paralleler Hybridantrieb

Der Elektromotor beim parallelen Hybrid kann kleiner ausfallen, da er hauptsächlich zur Unterstützung des Verbrennungsmotors dient (Abbildung 2.2, Mitte). In der konventionellen Konfiguration werden die abgegebenen Momente der beiden Motoren summiert, es sind aber auch Zugkraft- und Drehzahladditionen möglich. Weitere Funktionen wie Boosten, Rekuperieren, Impulsstart und in manchen Konfigurationen auch rein elektrisches Fahren erhöhen die ökonomischen Vorteile dieses Antriebskonzepts.

Die elektrische Maschine beim Konzept des Parallelhybridfahrzeugs kann sowohl generatorisch als auch motorisch arbeiten. Die direkte Verbindung des Verbrennungsmotors zu den Antriebsrädern bietet vor allem bei höheren Geschwindigkeiten Vorteile im Kraftstoffverbrauch gegenüber dem seriellen Hybridfahrzeug. Bei geringeren Drehzahlen wirkt sich diese Struktur jedoch nachteilig auf den Verbrauch und somit die Emissionen aus [11].

2.5.4 Leistungsverzweigter Hybridantrieb

Gibt es die Möglichkeit bei der Konfiguration des seriellen Hybridfahrzeugs eine Verbindung zwischen dem Verbrennungsmotor und den Antriebsrädern herzustellen, liegt ein leistungsverzweigter Hybridantrieb vor (Abbildung 2.2, unten). Diese Kopplung kann in Form einer einfachen Kupplung bis zum Einsatz eines Planetengetriebes reichen. Diese Konfiguration vereint die Vorteile von seriellem und parallelem Antrieb, demgegenüber steht jedoch ein weit komplexerer technischer Aufbau und eine aufwendigere Regelungsstrategie [11].

Der Toyota Prius der dritten Generation ist als leistungsverzweigter Hybrid ausgeführt und auch als Plug-In Variante erhältlich. Die Verbindung der Motoren mit dem Antriebsstrang erfolgt über ein von Toyota entwickeltes Hybridantriebssystem, dem Hybrid Synergy Drive (HSD), welchem das Konzept eines Planetengetriebes zugrunde liegt [1].

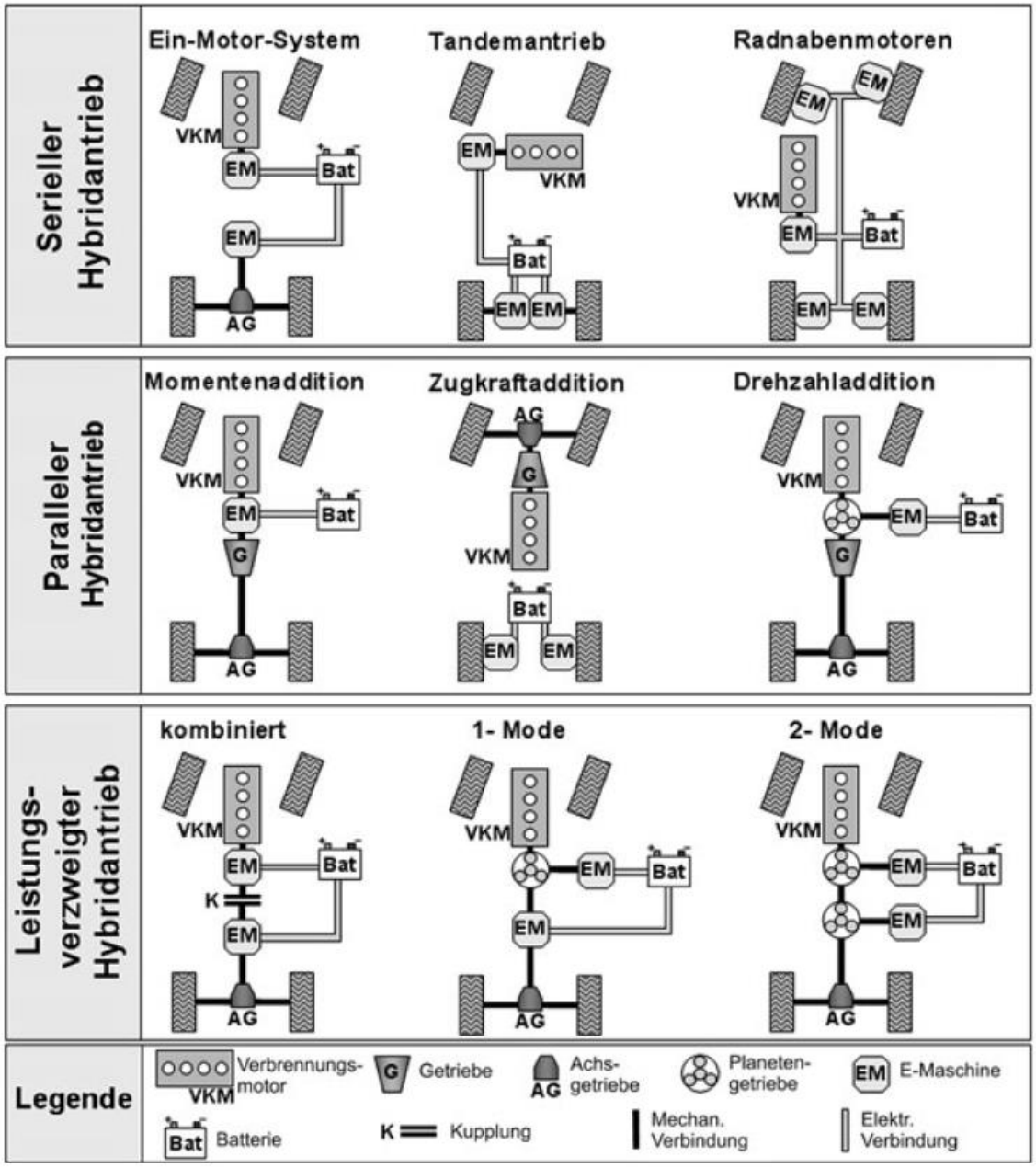

Abbildung 2.2: Strukturen von Hybridfahrzeugen [11]

2.6 Interne Schleifen

Bei einer Co-Simulation können direkte wechselseitige Abhängigkeiten zweier Modelle auftreten. Es sind prinzipiell drei Möglichkeiten der Simulationsreihenfolge für eine gleiche Makroschrittweite der Subsysteme möglich. Als Beispiel wird ein Regler mit Strecke (dynamisches System) gezeigt (Abbildung 2.3), welche in starker Abhängigkeit zueinander stehen. In Abbildung 2.3a wird zuerst die Strecke und anschließend der Regler sequentiell berechnet. Beim ersten Zeitschritt müssen die Eingangsgrößen der Strecke geschätzt werden, der Regler arbeitet anschließend schon mit den interpolierten Ergebnissen der Strecke.

Vertauscht man die Reihenfolge wie in Abbildung 2.3b, so müssen die Eingangsgrößen des Reglers zu Beginn extrapoliert werden und die Strecke verwendet als Eingang die interpolierten Ergebnisse.

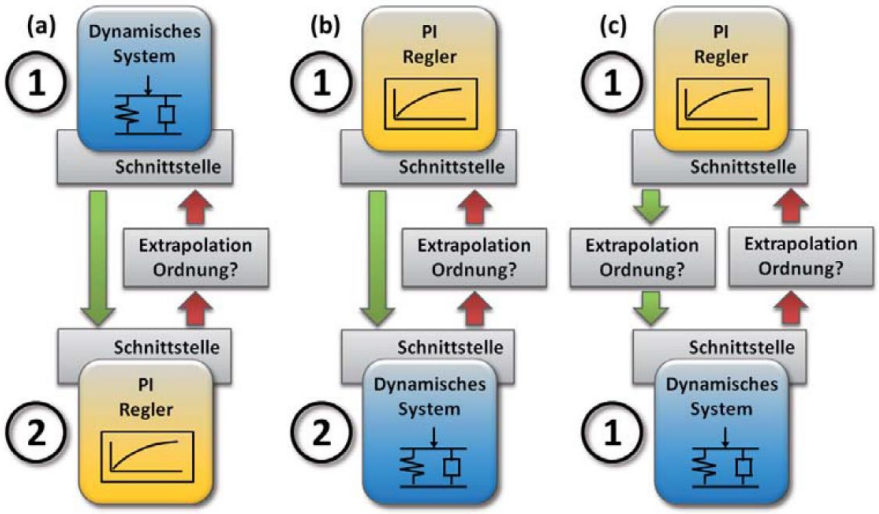

Abbildung 2.3: Subsysteme in einer Schleife [3]

Löst man die Modelle parallel (Abbildung 2.3c), so werden die Eingänge beider Subsysteme für den ersten Zeitschritt geschätzt. Die Simulationszeit wird dadurch zwar verkürzt, jedoch gibt es Auswirkungen auf die Stabilität und Genauigkeit der Simulation. Durch geeignet kleine Mikro- und Makroschrittweiten können die Simulationsergebnisse verbessert und über geeignete Koppelverfahren Unstetigkeiten in den Koppelsignalen vermieden werden [3].

2.7 Co-Simulation von gekoppelten Modellen

Ein wichtiger Parameter bei einer Co-Simulation ist die Reihenfolge, mit der die Subsysteme innerhalb eines Makroschritts berechnet werden. Grundsätzlich können die Modelle nacheinander, oder bei Mehrkernprozessoren auch parallel berechnet werden. Für die beiden folgenden Simulationsbeispiele wird das System aus Abbildung 2.3 mit einem Eingang und einem Ausgang verwendet.

2.7.1 Sequentielle Berechnung der Modelle

Werden die Modelle in Serie berechnet spricht man auch von “Gauß-Seidel” Typ (Abbildung 2.4). Der Abstand zwischen T_n und T_{n+1} entspricht einem Makrozeitschritt. Zum Zeitpunkt T_n wird zuerst das Subsystem 1 simuliert. Seine Eingangsgröße u_1 wird aus den in den vorherigen Simulationsschritten erhaltenen Werten extrapoliert (lineare Extrapolation von y_2). Am Ende des Makroschritts zum Zeitpunkt T_{n+1} wird das berechnete Ausgangssignal an das Subsystem 2 übermittelt, welches nun mit der Berechnung des Makroschritts zum Zeitpunkt T_n beginnt. Sein Eingangssignal u_2 kann aus dem Ergebnis von Subsystem 1, y_1, interpoliert werden. Hat nun das zweite System diesen Makroschritt fertig gerechnet, sendet es seine Ausgangsgrößen ans erste System und ein neuer Makroschritt kann beginnen.

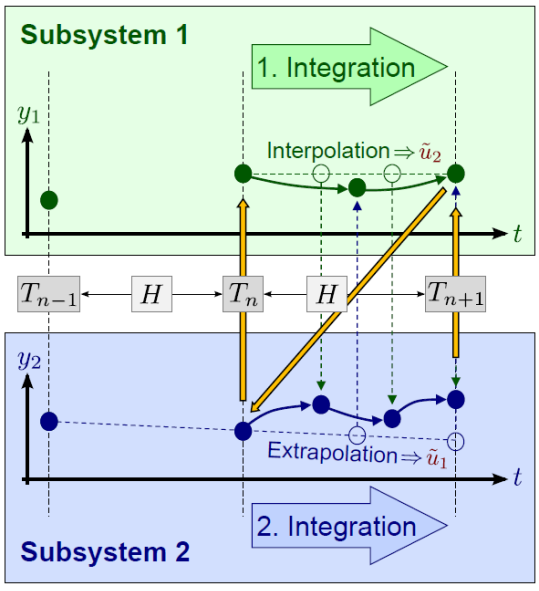

Abbildung 2.4: Subsysteme seriell simuliert [9]

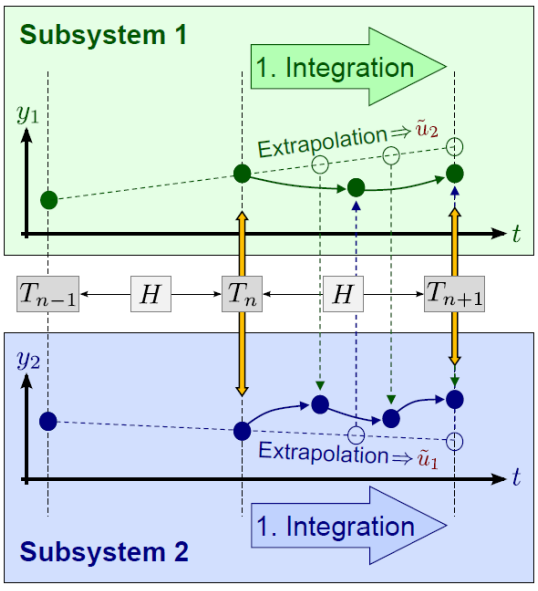

Abbildung 2.5: Subsysteme parallel simuliert [9]

Mit dieser Methode muss nur der Eingang für ein Modell extrapoliert werden, das zweite bekommt bereits die interpolierten Ergebnisse des ersten Subsystems. Es entsteht

jedoch eine erhöhte Zugriffszeit auf den Prozessor, welches bei einer Co-Simulation mit Echtzeitgeräten zu Problemen führen kann [9].

2.7.2 Parallele Berechnung der Modelle

Bei einer parallelen Ausführung, auch „Jacobi Typ“ genannt, beginnt die Simulation der beiden Subsysteme für jeden Makrozeitschritt gleichzeitig (Abbildung 2.5). Vor dem Start werden die jeweiligen Eingangsgrößen (u_1, u_2) der Modelle aktualisiert und während der Berechnung anhand der letzten Werte extrapoliert (für u_1 lineare Extrapolation von y_2; für u_2 lineare Extrapolation von y_1). Dies führt zu einem Fehler in den Ergebnissen beider Subsysteme, beschleunigt jedoch die gesamte Simulation [9].

2.8 Extrapolation

Mit Hilfe der Extrapolation wird ein Signalverlauf anhand vergangener Werte geschätzt. Die gängigen Verfahren sind die Extrapolation nullter Ordnung (zero-order-hold, ZOH), erster Ordnung (first-order-hold, FOH) und zweiter Ordnung (second-order-hold, SOH) [3], [12], [13].

Bei diesen sogenannten Lagrangen Polynomen wird ausgehend vom aktuellen Makrozeitschritt T_n ein Polynom vom Grad p durch die letzten Stützstellen von $y_i(T_n)$ bis $y_i(T_{n-p})$ gelegt, um den Wert zum nächsten Makrozeitschritt $y_i(T_{n+1})$ zu berechnen (Abbildung 2.6).

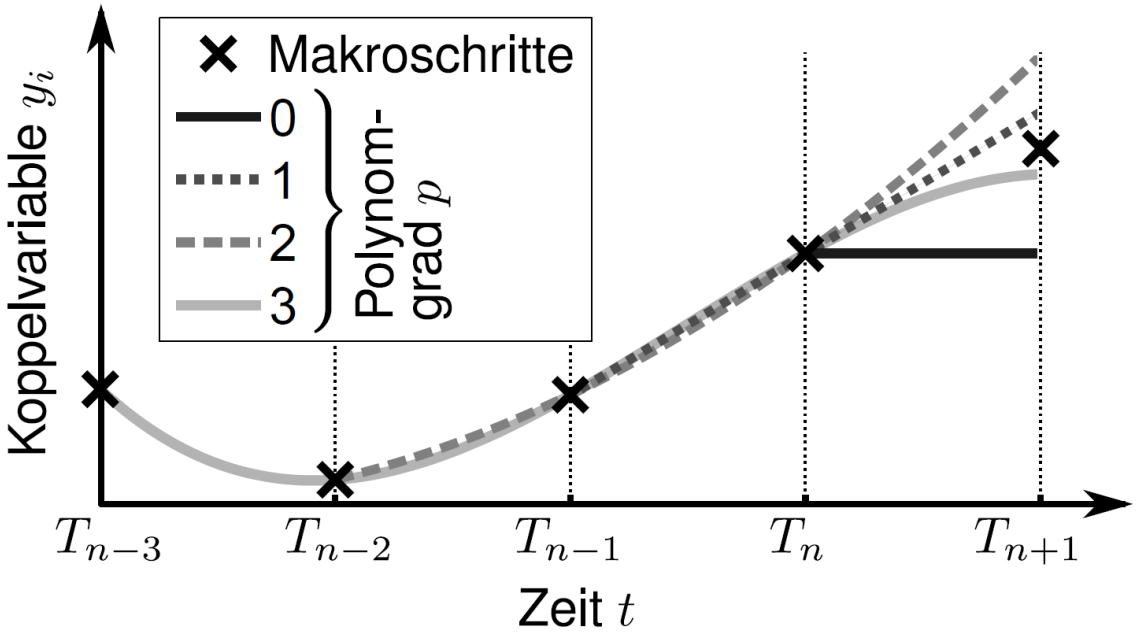

Abbildung 2.6: Extrapolationsverfahren mit Polynomen [9]

Da es keine exakten Methoden gibt, wird immer eine Abweichung vom tatsächlichen Wert vorhanden sein. Diese führt im nächsten Zeitschritt zu Unstetigkeiten, wenn der extrapolierte Wert als Stützstelle für die neue Extrapolation herangezogen wird [9].

2.8.1 Extrapolation erster Ordnung

Eine nähere Betrachtung verdient die Extrapolation nullter Ordnung, da sie in industriellen Anwendungen aufgrund ihrer Einfachheit häufiger als die anderen zuvor erwähnten Methoden eingesetzt wird. In Abbildung 2.7 ist das Verfahren bildlich dargestellt. Das extrapolierte Signal (rot) gibt konstant die Werte des Koppelsignals (blau) zu den Synchronisierungszeitpunkten wider.

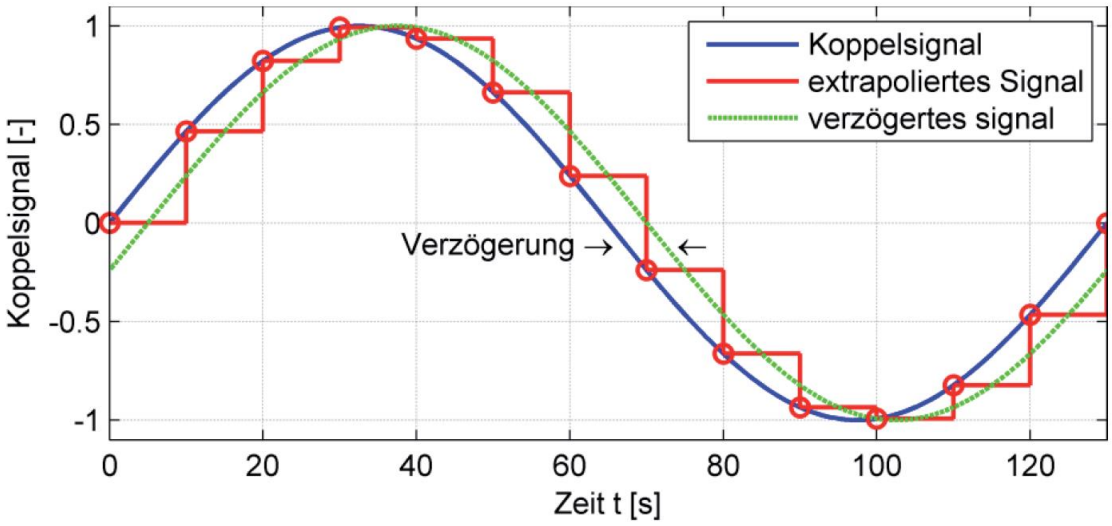

Abbildung 2.7: Koppelsignal mit extrapoliertem und verzögertem Signal [3]

Das resultierende Signal (grün) ist zeitverzögert und stellt eine Totzeit im System dar. Diese Totzeit beschränkt den Bereich der Makroschrittweite und hat Einfluss auf das Stabilitätsverhalten von geregelten Systemen [3].

2.8.2 NEPCE

Das Koppelverfahren „nearly energy preserving coupling element“ (NEPCE) kann den Extrapolationsfehler und die Totzeit fast eliminieren und führt zu einer erhöhten Genauigkeit sowie kürzeren Simulationszeit [3].

Die Effizienz liegt in der Annahme, die Koppelgrößen seien langsame zeitabhängige Funktionen. Diese Eigenschaft erreicht man durch eine geeignete Wahl der Makroschrittweite. Es wird mittels der Bond-Graphen-Theorie der Koppelfehler als Energiedifferenz in einem Zeitschritt ausgedrückt und für den nächsten

Simulationsschritt wird die Eingangsgröße zur Vermeidung einer Energiedifferenz optimiert [10].

Koppelt man diese Methode mit einer Energiedifferenzbasierten Makroschrittweitenberechnung (ECCO), so erzielt man noch bessere Ergebnisse in Genauigkeit und Rechenzeit [14].

2.9 Datenaustausch zwischen Systemen

Es gibt zwei generelle Arten von Koppelverfahren mehrerer Systeme:

- Bei der Iterativen-Kopplung kann eine Fehlerschranke vorgegeben werden und für den aktuellen Makrozeitschritt wird das Subsystem mehrmals berechnet, bis diese unterschritten wird. Beim ersten Schritt kommt es zu einer Extrapolation der Koppelgrößen. Für jede Wiederholung des Makroschritts müssen Parameter des Systems zurückgesetzt werden, wie etwa die Simulationszeit oder Systemzustände. Da nur wenige Modellierungsumgebungen diese Vorgaben erfüllen, ist die iterative Methode nur beschränkt anwendbar [3].
- Wird jedes Teilsystem in jedem Makroschritt nur einmal berechnet, spricht man von Nicht-Iterativer-Kopplung. Es gibt einen größeren Extrapolationsfehler in jedem Schritt, da nur die letzten Größen zur Extrapolation herangezogen werden. Die einzelnen Subsysteme können parallel oder sequentiell berechnet werden [3].

Erstere Methode wird auch „implizit“ genannt, da eine Makroschrittwiederholung notwendig ist. Im Gegensatz dazu ist die Nichtiterative Kopplung explizit [9].

2.10 Makro- und Mikroschritte

Ein wichtiges Merkmal einer Co-Simulation ist die unabhängige Lösung der Teilsysteme mit ihren eigenen Domänenspezifischen Solvern und Zeitschritten. Diese Zeitschritte werden als Mikrozeitschritt (δT) bezeichnet (Abbildung 2.8). In den Modellen kann die Schrittweite fest („fixed“) oder variabel („variable“) eingestellt werden. Innerhalb eines Mikrozeitschritts wird ein System einmal simuliert.

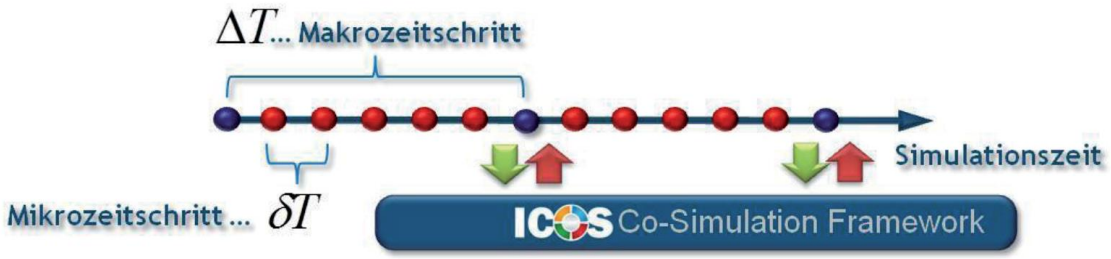

Abbildung 2.8: Mikro- und Makrozeitschritte [3]

Der Datenaustausch zwischen den Modellen findet zu bestimmten Zeitpunkten statt, am Ende eines sogenannten Makrozeitschrittes (ΔT) oder Synchronisationszeitpunkt. Innerhalb eines Co-Simulations-System können die Subsysteme unterschiedliche Makroschrittweiten haben [3].

Ein mechatronisches System besitzt ursächlich unterschiedliche Dynamiken. So ist etwa das elektrische System schneller und benötigt öfters neue Eingangswerte als ein vergleichsweise träges mechanisches. Die Signale zwischen den Synchronisationszeitpunkten werden dann extrapoliert.

Zur Veranschaulichung wird die Vorgehensweise bei einem elektromechanischen System beschrieben. Ein mechanisches System besitzt eine Mikroschrittweite von 10ms und eine Makroschrittweite von 100ms. Die Schrittweiten des gekoppelten elektrischen Modells sind 1ms und 10ms. Würden die Ausgangssignale des mechanischen nun mit ZOH extrapoliert werden, bekäme das elektrische zehn Mal den gleichen Wert am Eingang. Um dies zu vermeiden, wird mit unterschiedlichen Ansätzen die Extrapolation verbessert.

Zu den Synchronisationszeitpunkten werden alle Modelle gestoppt, um einen konsistenten Datenaustausch zu gewährleisten. Dies funktioniert nicht, wenn in die Co-Simulation ein Echtzeitsystem eingebunden wird. Für diesen Fall müssen alle Modelle schneller als Echtzeit rechnen und zusätzlich muss ausreichend Zeit für die Synchronisation zur Verfügung stehen.

Verwendet man eine variable Mikroschrittweitensteuerung in den Modellen, so darf die obere Grenzzeit die dem Modell zugeordnete Makroschrittweite nicht übersteigen.

2.11 Echtzeit-Co-Simulation

Werden in einer Co-Simulation Echtzeitgeräte eingebunden, etwa über ein Bussystem, ändern sich die Anforderungen an die beteiligten Systeme und Solver. Jedes Modell in

der Co-Simulation muss in Echtzeit berechnet werden und seine Ergebnisse zeitgerecht zur Verfügung stellen. Bei komplexen Modellen wird ein iteratives Lösungsverfahren nicht implementierbar sein und es ist auch auf Kopplungsfehler Rücksicht zu nehmen. Eine große Schwierigkeit bilden geschlossene Regelkreise hinsichtlich Stabilität und Konvergenz, da sich Latenzzeiten durch Synchronisations- und Berechnungszeiten ergeben. Durch die Vernetzung von Hardwarekomponenten, die beispielsweise aus Steuergeräten oder Sensoren bestehen, mit rechnergestützten Simulationsmodellen können HiL-Tests aufgebaut werden [15].

2.11.1 Echtzeitbedingungen

Bei einer Kommunikation zwischen einem PC und einem Echtzeitgerät sind verschiedene Herausforderungen zu meistern, besonders bei der Verwendung eines Standard-Windows-Betriebssystems ist Vorsicht geboten. Es gibt Erweiterungen für Windows 7, welche eine echtzeitfähige Schnittstelle garantieren, oder man verwendet ein Echtzeitbetriebssystem von Linux um Laufzeitschwierigkeiten möglichst zu umgehen. Bei dieser Arbeit sorgte das Co-Simulationsprogramm für die Einhaltung dieser Bedingungen.

Ein System wird als echtzeitfähig bezeichnet, wenn sowohl die zeitliche, als auch die logische Fehlerfreiheit seiner Funktion gegeben ist. Unter der Antwortzeit eines Systems versteht man die Zeitdauer zwischen anlegen aller Eingangsgrößen, bis das Ergebnis an den Ausgängen verfügbar ist. Die Einteilung von Echtzeitsystemen erfolgt nach der Art der Konsequenzen (Abbildung 2.9) bei verpasster Deadline [16].

SOFT REALTIME

Die Qualität eines Ergebnisses nach Verstreichen der Frist sinkt mit der Antwortzeit (Abbildung 2.9, links). Dies gilt etwa bei einer Benutzeroberfläche mit der dem Anwender Information so schnell wie möglich angeboten werden soll. Jedoch werden kurze Wartezeiten die Verwendbarkeit des Systems nicht stören [16].

FIRM REALTIME

Gelegentliches Verpassen der Frist ist tolerierbar, jedoch verschlechtern Versäumnisse die Systemqualität und können bei oftmaligem Auftreten zu Systemversagen führen (Abbildung 2.9, Mitte). Ein nicht fristgerechtes Ergebnis ist wertlos und kann nicht mehr verwendet werden, ein Beispiel wäre ein Videokonferenzsystem, bei dem vereinzelte Paketverluste geduldet werden können [16].

HARD REALTIME

Die Frist für die Antwort darf nicht überschritten werden, da dies zu einem Ausfall des Systems führt (Abbildung 2.9, rechts). Harte Echtzeitsysteme werden eingesetzt, wenn ein Verpassen einer Deadline zu großem Schaden führt, wie etwa die Verletzung einer Person oder die Beschädigung von Maschinen und Ausrüstung. Diese Anlagen arbeiten meist direkt mit der physikalischen Hardware zusammen, ein Beispiel ist das Auslösesystem für einen Insassenairbag eines Fahrzeugs [16].

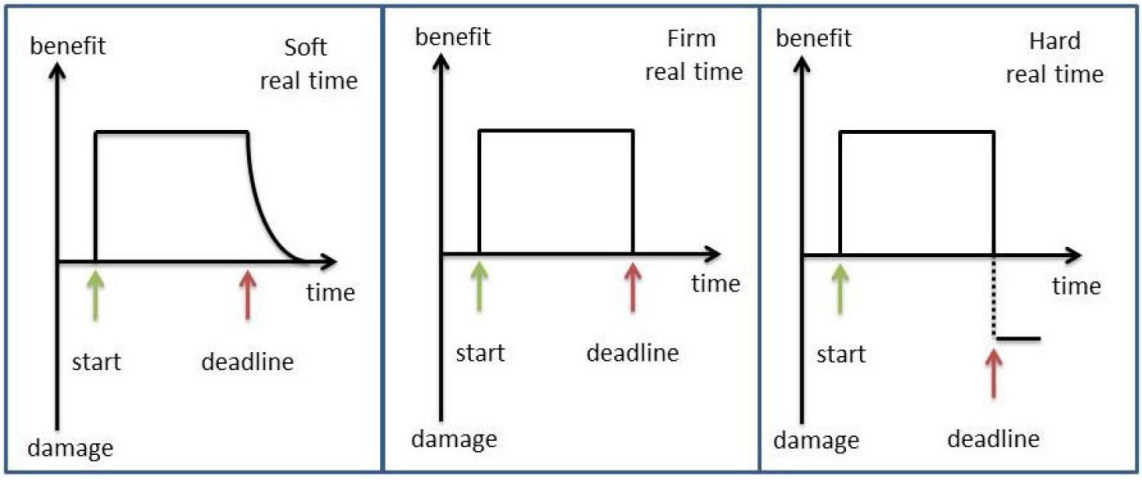

Abbildung 2.9: Echtzeitbedingungen mit den Konsequenzen bei Deadline-Überschreitung [16]

2.11.2 Bussysteme im Fahrzeug

Unter einem Bussystem versteht man ein Kommunikationsnetzwerk mehrerer Teilnehmer. Die Datenübertragung zwischen ihnen kann dabei mit Lichtwellenleiter, Drahtlos oder elektrisch über Eindraht- oder Mehrdrahtleitungen stattfinden.

Es gibt viele Systeme vom Buszugriffsverfahren, welche die Berechtigung der Teilnehmer, eine Nachricht zu schicken, regeln. Bei den zufälligen Verfahren warten die Teilnehmer bis der Bus frei ist und starten dann mit dem Senden ihrer Nachricht. Beginnen zwei gleichzeitig mit einer Übertragung, gibt es eine Kollision am Bus. Je nach Verfahren wird entweder das Senden abgebrochen und erst nach einer „Wartezeit“ bei freiem Bus wieder gestartet, oder es gibt am Beginn einer Nachricht eine Arbitrierungsphase, in der sich die Botschaft mit höherer Priorität durchsetzt und unterbrechungsfrei übermittelt wird. Bei deterministischen Verfahren existiert ein Master, der den Buszugriff vorbestimmt steuert. Die Slaves werden token- oder zeitgesteuert aufgefordert ihre Nachrichten zu senden.

Die Topologie eines Busses kann abhängig vom Buszugriffsverfahren sein. Die häufigsten Strukturen sind in Abbildung 2.10 dargestellt. Eine Stern-Topologie (a) ist ein klassisches Beispiel eines zentral gesteuerten Systems. Weitere Möglichkeiten der Anordnung der Teilnehmer wäre die Bus- (b) und Ringtopologie (c).

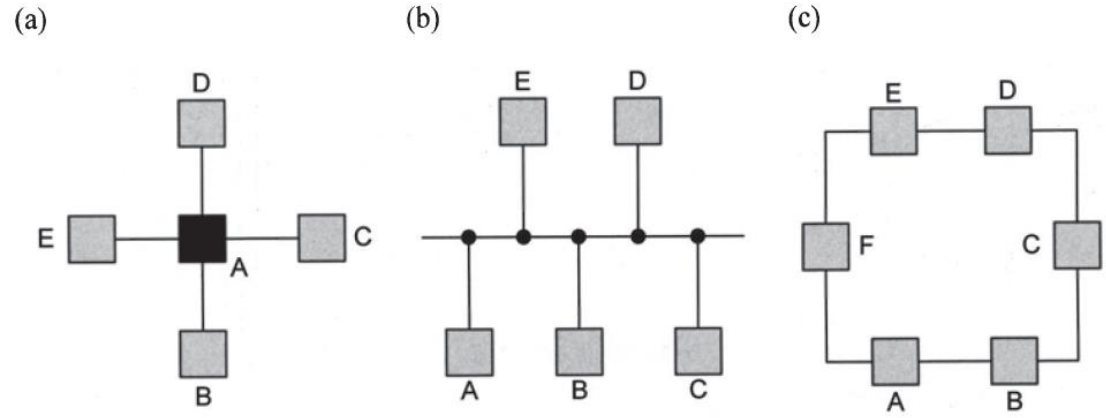

Abbildung 2.10: Topologien von Bussystemen [17]

Eine Nachricht besteht neben den eigentlichen Nutzdaten auch aus einem Identifier, der einen Teilnehmer direkt adressiert, oder Auskunft über den Inhalt einer Nachricht gibt und die Entscheidung des Empfangs den Zuhörern am Bus überlässt. Weitere zusätzliche Daten in einer Nachricht befassen sich mit der Datensicherheit. Es gibt unterschiedlich aufwendige Verfahren, angefangen vom einfachen Parity Check bis zu den zyklischen Blocksicherungsverfahren (CRC).

Im Fahrzeug werden aufgrund von Anforderungen an Sicherheit, Bandbreite, Determinismus und Kosten verschiedene Bussysteme verwendet. Die wichtigsten werden kurz erläutert [17].

CAN Bus

Der „Controller Area Network“ Bus wurde 1986 von der Firma Bosch entwickelt und hat sich heute als Standard-Bussystem in Personen und Nutzfahrzeugen etabliert. Er dient hauptsächlich zur Kommunikation der Steuergeräte untereinander, wobei meist mehrere CAN Busse in einem Fahrzeug zum Einsatz kommen. So kann etwa ein Hybridsteuergerät über einen CAN Bus mit dem Getriebesteuergerät, mit einem weiteren mit dem Batteriesteuergerät und mit einem dritten mit der Leistungselektronik des DC/DC Konverters kommunizieren.

Jede Nachricht ist durch einen eindeutigen Identifier gekennzeichnet und die Busteilnehmer entscheiden selbst über die Übernahme der Nachricht. Bei der eingesetzten Bustopologie (Abbildung 2.10b) sind bis zu 110 Teilnehmer möglich. Die Übertragungsrate ist stark abhängig von der Leitungslänge und geht von 1 Mbit/s bei kurzen Leitungen (<40m) bis 125 kbit/s bei etwa 500m [18], bei längeren Übertragungswegen nimmt die Datenrate weiter ab.

Der physikalische Aufbau besteht aus einem Leitungspaar, welches ein differentielles Signal überträgt. Es ist ein jeweils gegenphasiges Spannungsniveau auf den Leitungen, wie in Abbildung 2.11 für den High-Speed-CAN dargestellt.

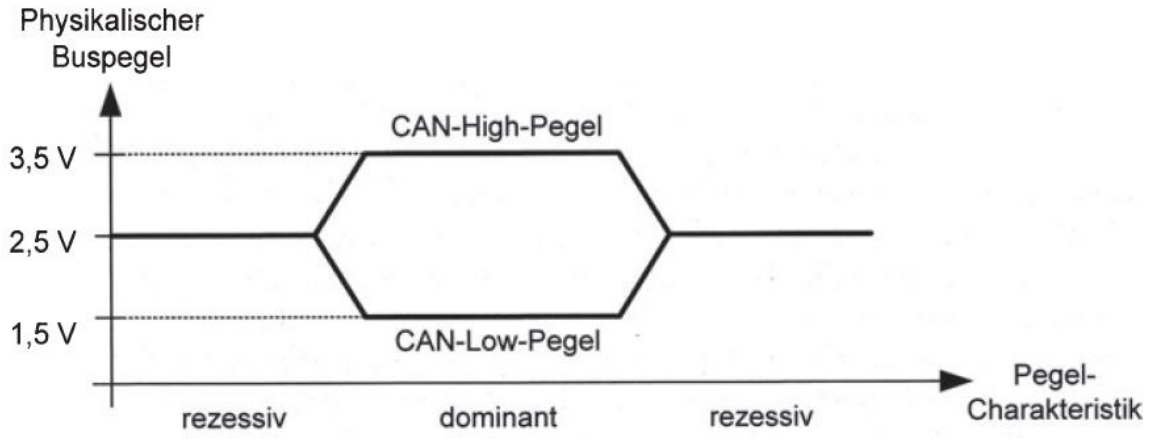

Abbildung 2.11: CAN Buspegel [17]

Der Vorteil bei differentiellen Signalen liegt in der Sicherheit gegen eingestreute Gleichtaktstörungen auf beiden Leitungen, da nur die Differenz der Pegel wichtig ist.

In Abbildung 2.12 ist ein beispielhafter Aufbau eines CAN Netzwerk zu sehen. Jeder Teilnehmer ist mit beiden Leitungen verbunden und am Ende dieser muss ein Abschlusswiderstand von $120\,\Omega$ zur Vermeidung von Reflexionen auf den Leitungen angebracht werden.

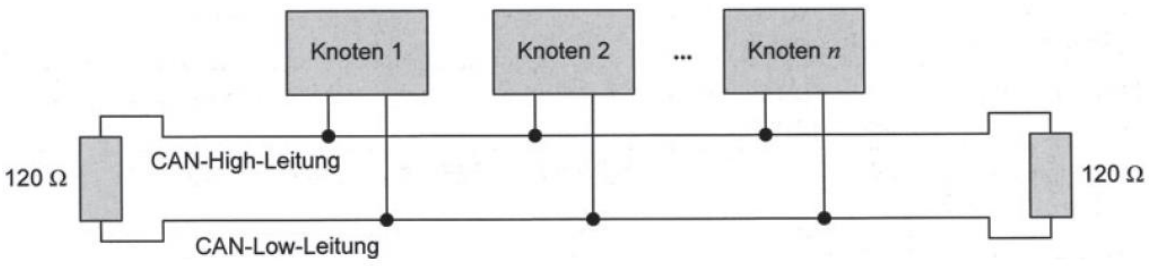

Abbildung 2.12: CAN Bus Topologie mit Abschlusswiderständen [17]

Aufgrund seiner bedingten Echtzeitfähigkeit über priorisierte Nachrichten, der Einfachheit des Aufbaus, der großen Zahl an möglichen Teilnehmern und der vielen Möglichkeiten zur Datensicherheit wird der CAN Bus in Fahrzeugen eingesetzt [17].

LIN Bus

Der „Local Interconnect Network“ Bus wurde 1998 von mehreren Herstellern entwickelt, um eine günstige Vernetzung von mechatronischen Elementen mit geringen Datenraten bereitzustellen. Es wird ein Master-Slave Verfahren eingesetzt bei dem die Kommunikation über „Tasks“ stattfindet. Wie schon beim CAN Bus, sind auch hier die Botschaften mit einem Nachrichtenidentifier ausgestattet.

Der Master schickt einen „Header“, in dem hauptsächlich der Identifier steht und der für diese Nachricht zuständige Slave antwortet mit einem „Response“ (Abbildung 2.13). Zusammen ergibt sich so ein „Frame“.

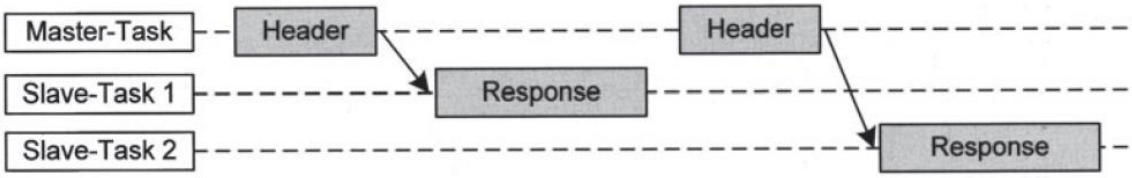

Abbildung 2.13: LIN Bus Kommunikation zwischen Master und Slave [17]

Die Nutzdaten im „Response“ bestehen aus maximal 8 Bytes. Es wird eine Eindrahtleitung verwendet, wobei sich der Signalpegel vom Bordnetz ableitet. Die Signale können über Standardschnittstellen wie SCI gesendet werden. Aufgrund der Eindrahtleitung und der auf die Fahrzeugmasse bezogene Übertragung sollten die Bandbreite maximal 19200 Bit/s betragen [17].

Die einfache Vernetzung von Komponenten wie der Niederspannungsbatterieüberwachung mit einem Steuergerät oder auch der Bauteile innerhalb einer Fahrzeugtür oder –sitz gehören zu den Einsatzgebieten.

FLEXRAY

Das Flexray Konsortium wurde 1999 gegründet um mit den steigenden Anforderungen an Bandbreite und Übertragungssicherheit Schritt zu halten. Es wurde ein deterministischer und fehlertoleranter Bus entwickelt, der noch Möglichkeiten für eine zukünftige Erweiterung bietet.

Es sind Stern- und Busstrukturen, sowie Kombinationen aus beiden möglich, sogar mit mehreren Masterelementen. In der Ausführung von Zwei-Kanal-Systemen ist die redundante Übertragung von Botschaften möglich. Ein Kommunikationszyklus enthält ein dynamisches und statisches Segment. Mit ersterem ist eine prioritätsgesteuerte Datenübertragung möglich und kann die für den Antriebsstrang notwendigen Botschaften kurbelwellensynchron übertragen. Mit dem statischen Segment können die Busteilnehmer in festen Zeitfenstern ihre Signale deterministisch senden. Dafür müssen sich alle Busknoten auf die gleiche Zeitbasis beziehen, was über aufwendige Korrekturverfahren aufgrund von Toleranzen oder Alterung sichergestellt wird.

Eine Nachricht besteht aus drei Segmenten (Abbildung 2.14). In der ersten befindet sich die Identifikationsnummer der Nachricht zusammen mit ihrer Prüfsumme. Im zweiten Segment sind die Nutzdaten und im letzten die Prüfsummen der Nutzdaten.

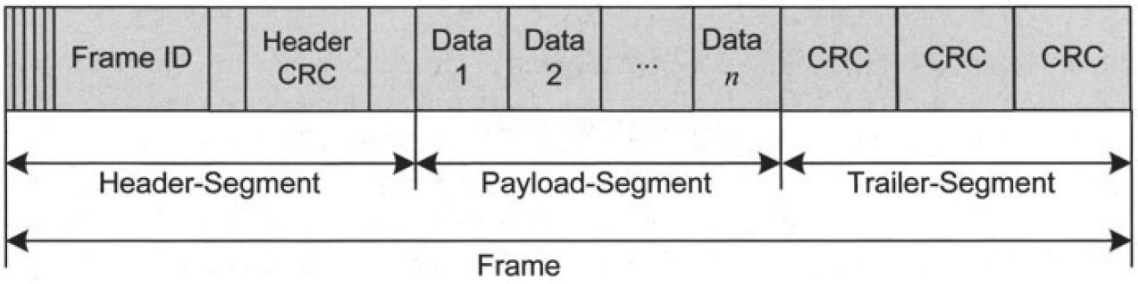

Abbildung 2.14: Flexray Nachrichtenformat [17]

Da Flexray als ein Protokoll gesehen werden kann, können optische und elektrische Übertragungsmedien eingesetzt werden. Die elektrische Signalübertragung erfolgt ähnlich wie beim CAN Bus über differentielle Spannungssignale um eine Leerlaufspannung von 2,5V. Es sind Datenraten bis 10 Mbit/s möglich [17].

3 Eingesetzte Programme

Die Erarbeitung dieser Masterarbeit erforderte den Einsatz folgender Software Tools:

- MATLAB/Simulink (R2013b 32 Bit)
- AVLab (v2.8.4)
- Model.CONNECT (v2016 ISO88)
- PEAK Konfigurationsprogramme

Im folgenden Kapitel wird ein Überblick über diese gegeben und nur auf die verwendeten Funktionen näher eingegangen.

3.1 MATLAB

Die MATLAB Plattform ist für die Lösung von technischen und wissenschaftlichen Problemen optimiert. Mit Hilfe der Matrix-basierten Sprache ist es intuitiv möglich, rechnergestützte Mathematik einfach auszudrücken. Dank implementierter Grafikfunktionen können Daten visualisiert werden und einige Toolboxen bieten Unterstützung für zahlreiche Problemstellungen an [19].

Es wurde mit einer 32 Bit Version von MATLAB R2013b gearbeitet. Das Co-Simulationsprogramm unterstützte alle Modelle die mit der MATLAB Version R2009 und jünger erstellt wurden.

3.2 Simulink

Simulink ist eine Blockdiagrammumgebung für die Mehrdomänen-Simulation und Model-Based Design. Simulink unterstützt den Entwurf und die Simulation von Systemen und ermöglicht außerdem die automatische Codegenerierung und das kontinuierliche Testen und Verifizieren von Embedded Systems.

Simulink umfasst einen Grafikeditor, benutzerdefinierbare Blockbibliotheken und Solver für die Modellierung und Simulation von dynamischen Systemen. Simulink ist in MATLAB integriert, sodass MATLAB-Algorithmen in Modelle aufgenommen und Simulationsergebnisse wiederum in MATLAB analysiert und weiterverarbeitet werden können [20].

Die Software für das Hybridsteuergerät und das Plant Modell wurden in Simulink realisiert.

3.2.1 ICOS Schnittstelle

Die Kommunikation mit Simulink fremden Programmen erfolgte über *ICOS* Blöcke, welche Funktionen wie Eingang in sowie Ausgang aus Simulink innehaben (Abbildung

3.1). Über diese Schnittstelle ist es möglich, Signale über das Co-Simulationsprogramm Model.CONNECT mit anderen Simulationsprogrammen oder Hardwareschnittstellen auszutauschen. Fügt man diese Blöcke, welche nach der Model.CONNECT Installation in der Simulink Library erhältlich sind, in Simulink ein, so werden diese Ein- und Ausgänge bei Einbinden des Modells im Co-Simulationsprogramm sichtbar.

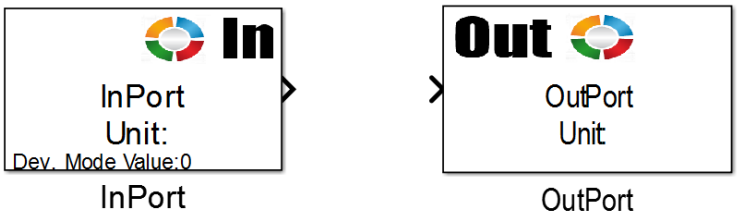

Abbildung 3.1: ICOS Blöcke in Simulink als Schnittstelle zu Model.CONNECT

Es ist zu beachten, dass nur Signale mit dem Datentyp *double* übertragen werden können, gegebenenfalls müssen somit die Datentypen der zu übertragenden Signale konvertiert werden. Die Einstellparameter dieser Blöcke beschränken sich auf den Signalnamen und aus der Dimension des Signals, letzterer ist größer eins bei der Verwendung von Vektoren.

3.2.2 Vektoren

In Kapitel 6 dieser Arbeit wird besonderes Augenmerk auf die Kommunikation zwischen Systemen mit vielen Schnittstellen gelegt, weil sie die Echtzeitfähigkeit der Gesamtsimulation beeinträchtigt. Ein Faktor dabei spielt die Synchronisation der Daten, die bei vielen Signalen erheblichen Einfluss auf die Laufzeit hat. Um viele Signale schnell zu übertragen kann man Vektoren verwenden.

Es gibt die Möglichkeit, mehrere Signale zu einem Bus zusammenzufassen und in einen Vektor zu packen. Der Vorteil dieses Aufbaus (Abbildung 3.2) liegt in der Verringerung der Rechenzeit. Für jeden *ICOS Block* wird eine S-Funktion in Simulink ausgeführt, die zusätzlich zur eigentlichen Datenübertragung Zeit benötigt. Bei 300 Signalen wird dieser Mehraufwand durch den Einsatz von Vektoren drastisch gesenkt.

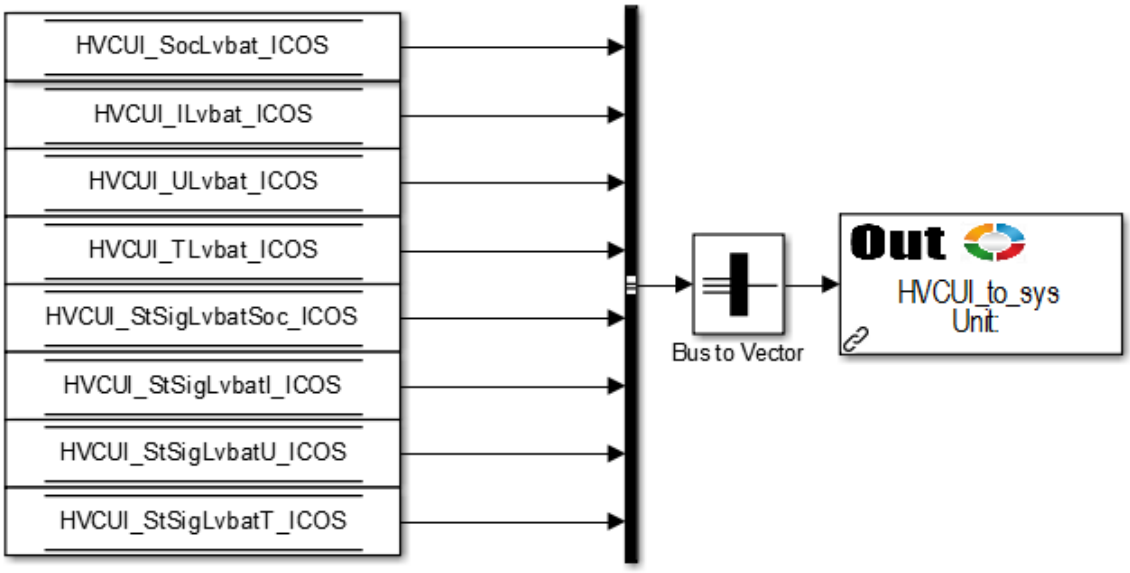

Abbildung 3.2: Mehrere Signale zu einem Vektor zusammengefasst

Der *Bus to Vektor-Block* ist ein Standard-Simulink Block. Er fasst Signale mit gleichen Datentypen zusammen, welche am Ausgang nur mehr in der Reihenfolge der Eingangssignale erkennbar sind, es geht jegliche Information über Signalnamen verloren.

Schickt man diese Daten in ein anderes Simulink Modell, so wird das Pendant wie in Abbildung 3.3 aufgebaut. Nach dem *ICOS-In-Block*, der Daten aus Model.CONNECT empfängt, wird eine Konvertierung von einem Vektor zu einem Bus benötigt. Dieser *vec2bus* ist in der *ICOS Library* zu finden und besteht aus einem *Demux*, welcher den Vektor aufspaltet und anschließend einem *Bus Creator*, der aus den Signalen einen Bus erstellt. Damit die Signale im Bus den richtigen Namen haben, muss im Bus Editor eine Busstruktur erstellt werden, welche dann im *vec2bus* geladen werden kann. Wird die Busstruktur als *.mat-Datei geladen und ein *Bus Selector* nachgeschalten, werden in diesem bereits die Bussignale angezeigt.

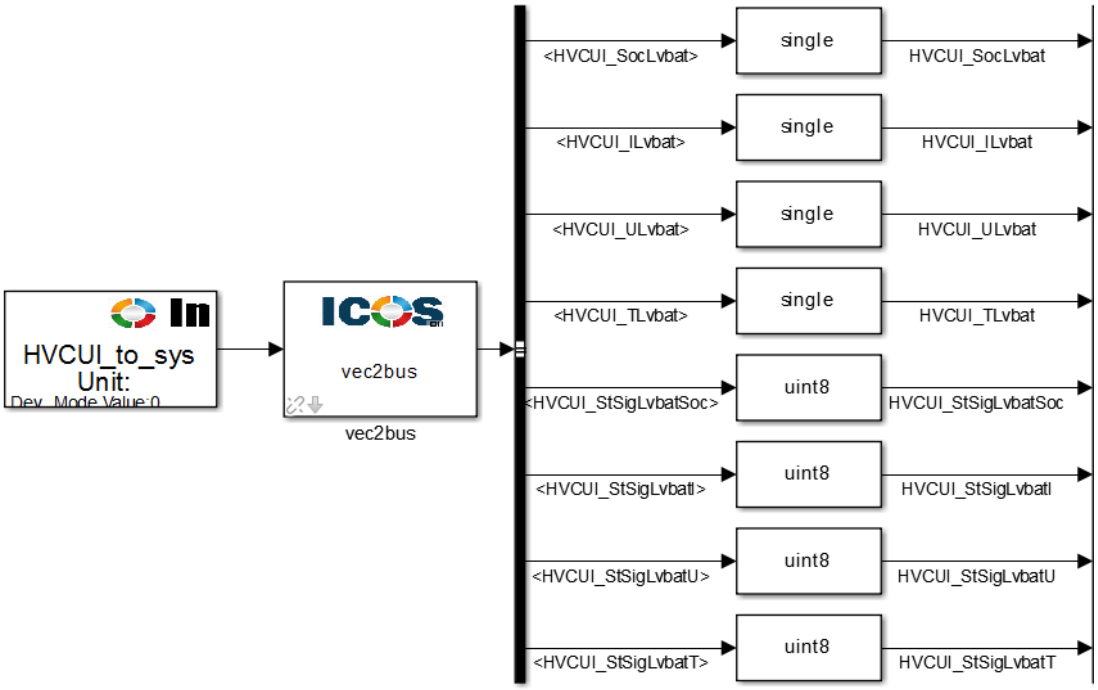

Abbildung 3.3: Aus einem Vektor werden wieder einzelne Signale

Da die innere Struktur des *vec2bus* Blocks für jedes System an die unterschiedliche Anzahl von Signalen angepasst wird, muss die Verlinkung zur *ICOS Library* abgebrochen werden.

3.3 AVLab

AVLab ist eine Eigenentwicklung von der Firma AVL List GmbH und gilt als Nachfolger des internen Entwicklungsprogramms PoET (Powertrainsoftware Engineering Tool). Es wird für die Entwicklung der Steuergeräte-Software eingesetzt. Verwendet wurde innerhalb der vorliegenden Masterarbeit AVLab v2.8.4.

Die Vorteile dieses Programms sind:

- Die Vereinigung mehrere Instrumente, welche die Entwicklungsstufen von Modellierung über Testen bis zur Codegenerierung unterstützen, um dadurch die Effizienz und Qualität bei der Funktionsentwicklung zu verbessern.
- Signaleigenschaften in Simulink können mit Hilfe eines Synchronisationstools mit ADD abgeglichen werden.
- Das zur Dokumentation und Versionsverwaltung verwendete Programm *Integrity* wird auch von AVLab unterstützt.

- *AVL Concerto* wird zur Visualisierung und Verifikation von Signalverläufen eingesetzt und kann direkt in AVLab verwendet werden.
- Tests wie MiL und SiL werden mit AVLab einfach durchführbar.
- Der Stimuli Generator ermöglicht es, verschiedene Daten zu laden, erstellen, bearbeiten und zu speichern sowie diese später als Anregungssignale für ein Simulink Modell oder für den *ETAS Labcar operator* zu verwenden.
- Zur Codegenerierung wird eine Toolbar zur Verfügung gestellt, die richtlinienkonforme Codegenerierung für *TargetLink Code* und *Embeddedcoder* bietet.
- Zur Modellentwicklung in Simulink wird eine einheitliche Struktur angeboten, welche auf der AVL Powertrain Software Library Komponenten Struktur basiert. Sie enthält ein Betriebssystem, welches realitätsnähere Simulationen erlaubt als eine reine Simulation in Simulink [21].

3.3.1 Simulink Struktur

Die von AVLab vorgegebene Struktur ist in Abbildung 3.4 ersichtlich.

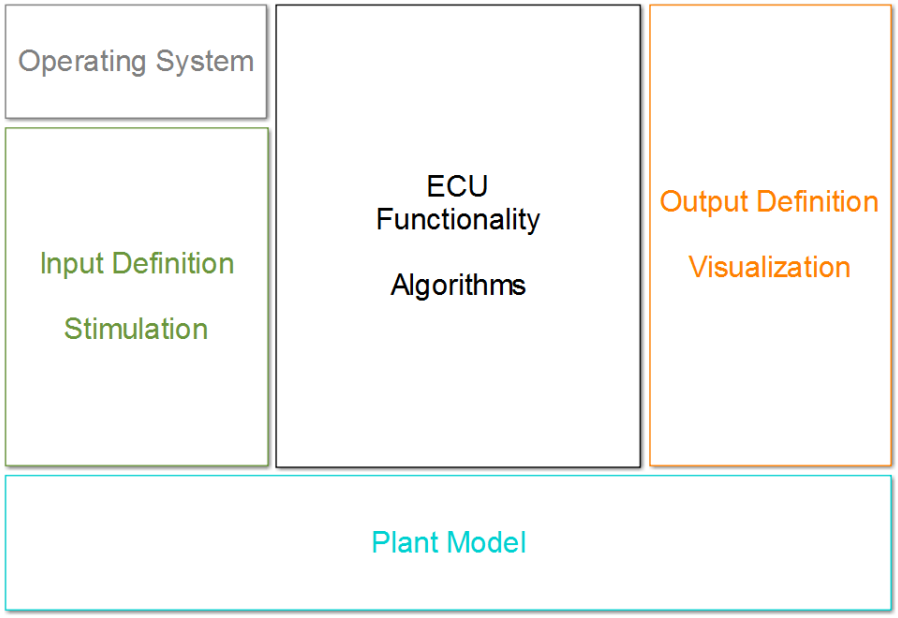

Abbildung 3.4: AVLab Standardstruktur in Simulink

Links oben befindet sich das Betriebssystem (*Operating System*) des Steuergeräts. Es gibt die für die Co-Simulation wichtige Zykluszeit von 10ms vor, in der auch das Hybrid-Steuergerät arbeitet. Sie stellt die untere Schranke für die Makroschrittweite des Modells dar. Darunter, im Block *Input Definition*, werden dann zur Laufzeit die Signale von AVLab

angehängt. An vordefinierten *DataStoreWrite* Blöcken werden Signale eingespeist, die in AVLab für den jeweiligen Testfall erstellt wurden. Im Block *Output Definition* könnte man Scopes oder Displays einfügen, diese Funktion wird aber nicht genutzt, da die gesamte Datenauswertung in AVLab über *Concerto* stattfindet. Das Modul in der Mitte, *ECU Functionality*, besteht aus vielen Ebenen. In der Schicht darunter befindet sich ein Subsystem, welches vom Betriebssystem mit der Zykluszeit angesprochen und demnach nur alle 10ms einmal ausgeführt und berechnet wird (Abbildung 3.5).

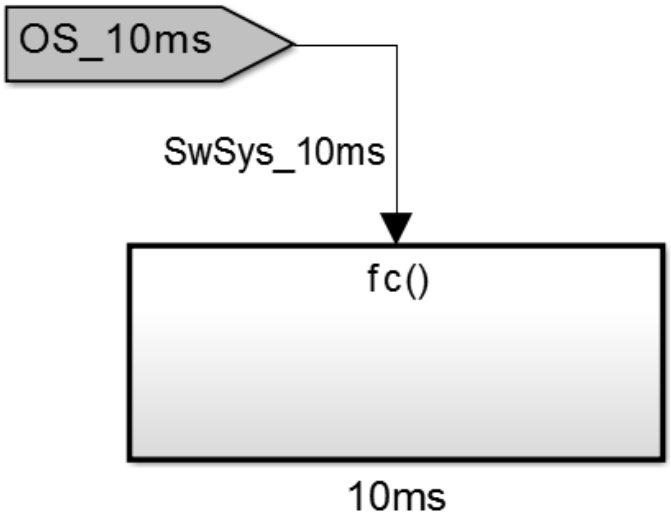

Abbildung 3.5: function-call subsystem vom Betriebssystem getriggert

Es handelt sich um ein *function-call subsystem*, das zyklisch wie eine Funktion aufgerufen wird.

Eine weitere Schicht darunter wird von *TargetLink* gebildet, und dient zur Generierung eines ausführbaren Programmcodes. Für diesen Block, wie in Abbildung 3.6, müssen

eigene Simulink Bibliotheken installiert werden. Zur einfacheren Handhabe der Ein- und Ausgänge werden sämtliche Signale zu einem Bus gebündelt.

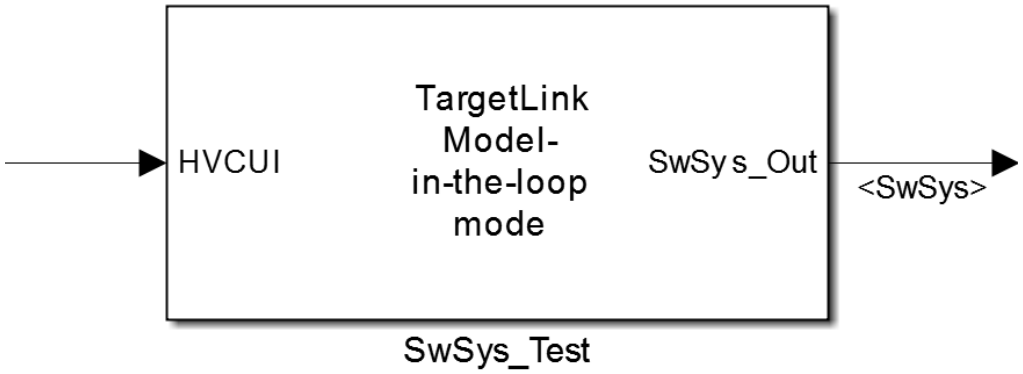

Abbildung 3.6: TargetLink Block

In diesem *TargetLink* Block befindet nun das Simulink Modell der Steuergerätsoftware, auch *SwSys* oder HCU, welche als *Referenced Model* eingebunden ist (Abbildung 3.7).

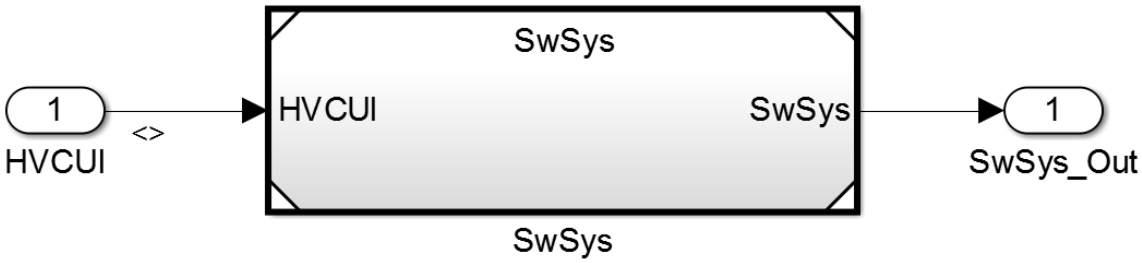

Abbildung 3.7: Einbindung der HCU Software

Der unterste Block in Abbildung 3.4 wird vom *Plant Model* gebildet. Hier wird das projektspezifische Modell des Fahrzeugs eingebunden. Es besteht im Wesentlichen aus einem AVL CRUISE Modell, in dem die Eigenschaften des Fahrzeugs abgebildet sind, sowie aus der Aufbereitung und Parametrierung der Signale.

3.3.2 Simulationsablauf

Die Funktionsweise von AVLab beim Start einer Simulation durchläuft mehrere Schritte und wird an dieser Stelle nun näher erläutert.

Erstellt man ein neues Projekt, so wird ein Simulink Modell mit dem schon beschriebenen Grundaufbau erstellt und unter dem Namen *XXX_Test* abgespeichert. Dieses Modell wird für alle Weiterentwicklungen verwendet und darf nicht umbenannt werden, da

andernfalls der Start aus AVLab nicht mehr möglich ist. Startet man nun eine Simulation, erstellt AVLab eine Kopie dieses Modells, benennt es in *XXX_miltest* um, fügt Speicherblöcke zur Signalaufzeichnung und für die Stimuli Signale ein, und schreibt Programmcode in die Callback Funktionen. Somit wird das Originalmodell nicht verändert.

Das bedeutet auch, dass für die Co-Simulation, welche aus AVLab heraus gestartet werden soll, die Kopie des Modells herangezogen werden muss.

3.4 Model.CONNECT

Die graphische Benutzeroberfläche des Co-Simulations-Programms wurde von der Firma AVL List GmbH entwickelt [22]. Es bietet zahlreiche Schnittstellen um Modelle aus verschiedenen Simulationsumgebungen einzubinden. Mit Hilfe von Model.CONNECT wird es möglich, Simulationsdaten aus unterschiedlichen technischen und physikalischen Disziplinen mit einander zu kombinieren. Auch kann eine Kommunikation mit Echtzeitanwendungen, wie Steuergeräten, einfach hergestellt werden.

Für folgenden Erwähnungen von Model.CONNECT wird die Abkürzung MC verwendet.

MC existiert seit dem Jahr 2015 und wird seither ständig weiterentwickelt. Jede Verbesserung im Programm wird als Beta-Version, den sogenannten ISOs, freigegeben. Das offiziell neueste Programm trägt den Namen Model.CONNECT v2015. Aufgrund von weiteren, für diese Arbeit unerlässlichen Entwicklungen wurde zuletzt mit MC v2016 (ISO88) gearbeitet

3.4.1 Aufsetzen einer Simulationsumgebung

Nach der Erstellung eines neuen Projekts werden die Simulationsmodelle über entsprechende Blöcke im *Topology*-Fenster eingebunden. Die Schnittstellen erscheinen dann in MC als Ports, welche man mit denen anderer Modelle einfach verbinden kann.

Für diese Arbeit wurden nur die Schnittstellen zu MATLAB und CAN Bussen benötigt, letztere wird über RealTime Blöcke (RT) hergestellt (Abbildung 3.8).

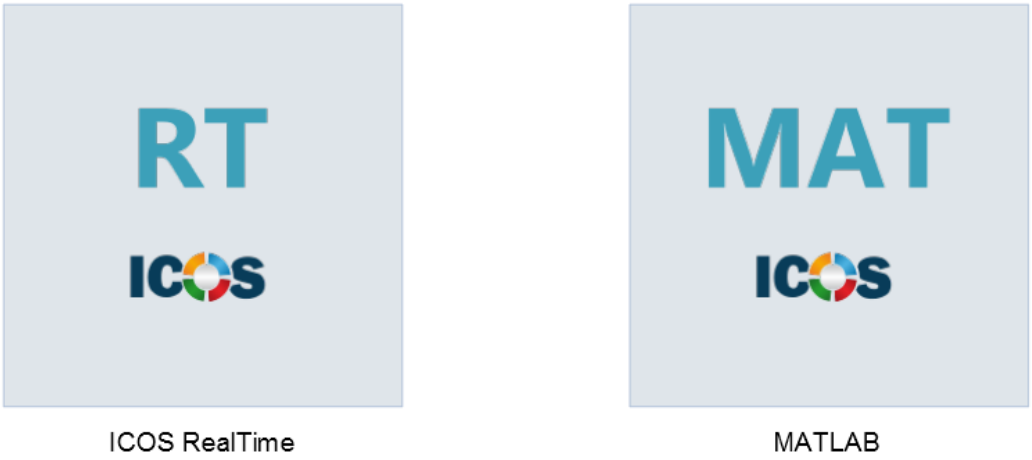

Abbildung 3.8: Verwendete Bausteine in MC

3.4.2 Einbindung von Simulink Modellen

Das Einbinden einer MATLAB/Simulink Schnittstelle erfordert eine Projekt-spezifische Parametrisierung. Im Folgenden sind die wichtigsten Schritte erläutert.

Der erste Parameter *Step size* definiert die Schrittweite, mit der die Eingänge des Modells aktualisiert und die Ausgänge abgefragt werden. Die *Step size* entspricht der Makroschrittweite und darf nicht kleiner als die Schrittweite im jeweiligen Modell sein.

Der Pfad zum Modell ist im Feld *Simulink model file* anzugeben. (Abbildung 3.9). Es können Simulink Modelle mit den Dateiendungen *.mdl und *.slx verwendet werden.

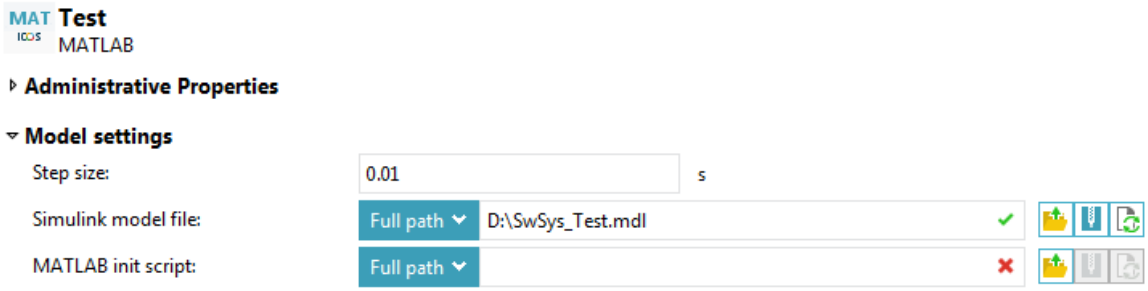

Abbildung 3.9: MC MATLAB Block Parameter 1

Als nächstes ist im Abschnitt *Execution settings* die Betriebssystem Plattform auszuwählen (Abbildung 3.10). Es wird zwar ein 64-Bit-Windows verwendet, aber da MATLAB als 32-Bit-Version installiert wurde, ist *Win32* auszuwählen.

Beim Punkt *MATLAB executable* wird der Pfad zur MATLAB Programmdatei angegeben. Wird MC von einem anderen Programm aus gestartet, im *Embedded Mode*, so muss

hier der *ICOS Port* mit dem Kommando *IcosPort:xxxx* angegeben werden, da dieses Modell schon in einem MATLAB Workspace geöffnet ist. Die Nummer des Ports wird beim Aktivieren dieser Funktion in MC eingestellt, mehr dazu in Kapitel 3.4.4. Bei den älteren MC Versionen (z.B. MC v2015 ISO91) musste beim *MATLAB startup mode* noch extra *Enable virtual machine* angegeben werden. Diese Funktion ist nun (MC v2016 ISO57) im Modus „Default“ voreingestellt. Sie wird benötigt, um die *TargetLink* Komponenten in den Modellen auszuführen.

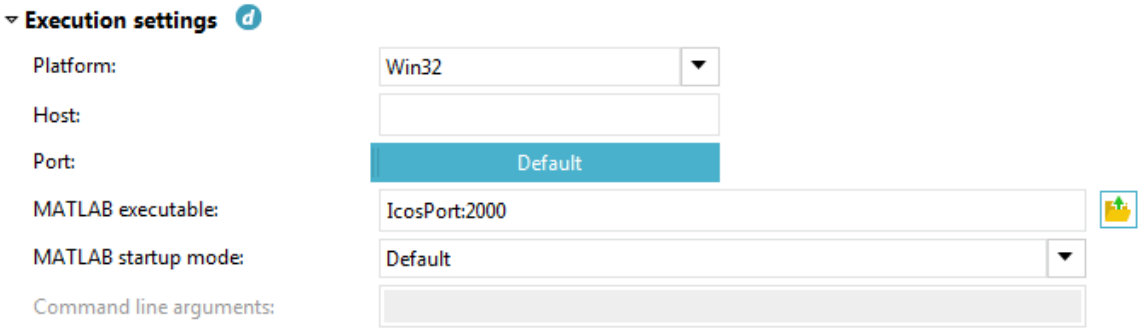

Abbildung 3.10: MC MATLAB Block Parameter 2

Man kann die Simulationen der einzelnen Modelle sequenziell oder parallel ablaufen lassen. Für den ersteren Betriebsmodi wird unter *Trigger sequence number* die Reihenfolge der Berechnung festgelegt (Abbildung 3.11).

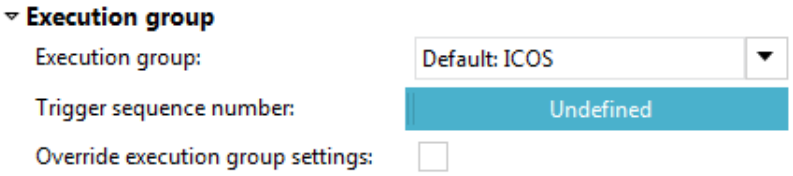

Abbildung 3.11: MC MATLAB Block Parameter 3

3.4.3 Einbindung eines CAN Busses mittels RealTime Block

Für die Kommunikation mit der Außenwelt über CAN ist in MC der Block *ICOS RealTime* zuständig. Zurzeit (September 2016) werden nur USB zu CAN Adapter von *PEAK Systems* und *Vector* unterstützt. Bei dieser Masterarbeit wurden PCAN-USB Geräte verwendet, da auch zur Simulation der Sensoren und Aktoren des Steuergeräts Geräte von *PEAK Systems* zum Einsatz kamen und diese wesentlich kostengünstiger als ihre Konkurrenzprodukte sind.

Für den RT-Block gibt es ähnliche Einstellungen wie vorhin, Schrittweite und Plattform, statt dem Modell wird jedoch auf ein *Project file* referenziert (Abbildung 3.12). In dieser *.ini-Datei wird die Eigenschaft des CAN Busses angegeben. Der genaue Aufbau dieser Datei ist im MC Handbuch angegeben [22].

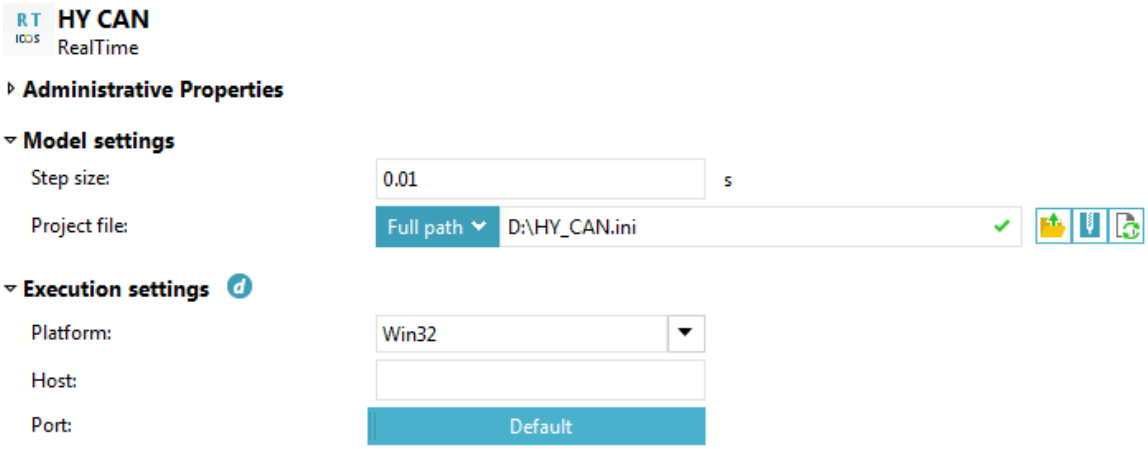

Abbildung 3.12: MC RT Block Parameter 1

Als erstes muss für jedes Signal ein Port erstellt werden, der anschließend in MC am RT-Block sichtbar wird, und die Extrapolationsart angegeben werden. Ohne genaueres Wissen über die Signalzusammenhänge sollte hier für den Anfang für alle Signale die Option Zero-Order-Hold ausgewählt werden, dafür wird nach dem Signalnamen die Nummer null eingetragen. Man definiert weiters die CAN Nachrichten und Signale und weist ihnen CAN Identifikationsnummern zu. Für ein jedes Signal sind auch seine Parameter, wie der ID der Nachricht, Startbit, Endbit, Faktor und Offset anzugeben. Trägt man alle Parameter händisch ein, kann nur die Byte-Ordnung Intel[1] verwendet werden.

Die Verwendung einer DBC-Datei ist möglich, und reduzierte die Einträge auf die Signalnamen und der ID der Nachrichten. Es muss dafür der Pfad dieser Datei angegeben werden und nun wird auch die Byte-Ordnung daraus übernommen (ab MC v2016).

Verwendet man mehrere PCAN-USB Adapter, so muss jedem eine eindeutige Identifikationsnummer zugewiesen werden. Diese hat im Bereich 0x51 bis 0x58 zu liegen und kann mit dem Programm *PCAN-View* eingestellt werden. Schließt man die PCAN-USB Adapter an einen Computer an, wird ihnen automatisch ein *Handle* zugewiesen. Mit dem PEAK-Programm *NetCfg32.exe* kann die Netzkonfiguration angesehen werden,

[1] Little Endian

darin sind alle Bauteile zur Buskonversion mit ihrer ID gelistet. Unter *Ansicht -> Handles anzeigen*, werden die *Handles* zu den einzelnen Geräten angezeigt. Die Nummer der *Handles* ist in MC fix mit dem CAN Kanal verknüpft, der in der *.ini-Datei beim Parameter *CAN-Channel* angegeben werden muss. Diese Zuordnung ist in Tabelle 1 angegeben.

Tabelle 1: Zuordnung der Handles an den CAN-Channel

Handle Nummer	CAN Kanal in der *.ini-Datei
16	81
15	82
14	83
13	84
12	85

Die Zuordnung der *Handles* zu den Geräten folgt dabei der zeitlichen Reihenfolge des Anschlusses. Der erste PCAN-USB Adapter bekommt den *Handle* 16 zugewiesen und muss folglich in der *.ini-Datei den *CAN-Channel* 81 zugewiesen bekommen. Das nächste angeschlossene Gerät bekommt den *Handle* 15 zugeteilt, das nächste dann 14, usw. In den MC Versionen ab v2016 ISO88 wird nun statt dem CAN Kanal direkt die *Handle* Nummer eingetragen. Diese IDs, sowie die ID der CAN Nachrichten sind in der *.ini-Datei in Dezimalschreibweise anzugeben.

Wählt man beim Punkt *Communication type to RT system* den CAN aus, ist beim *CAN driver path* die *PCANBasic.dll* für die jeweilige Betriebssystemarchitektur (32-Bit, 64-Bit), anzugeben (Abbildung 3.13). Diese Library für den PCAN USB Adapter findet man auf der Installations-CD oder Online bei *PEAK Systems*. Alternativ wird auch von MC eine Treiberdatei angeboten, welche man im Installationsverzeichnis unter dem Namen *CanApi2.dll* findet. Für den Einsatz mehrerer CAN Busse wurde die Verwendung letzterer empfohlen. Eine weitere Kommunikationsmöglichkeit wäre eine UDP Schnittstelle, welche eine einfache und schnelle, jedoch nicht zuverlässige Datenübertragung ermöglicht.

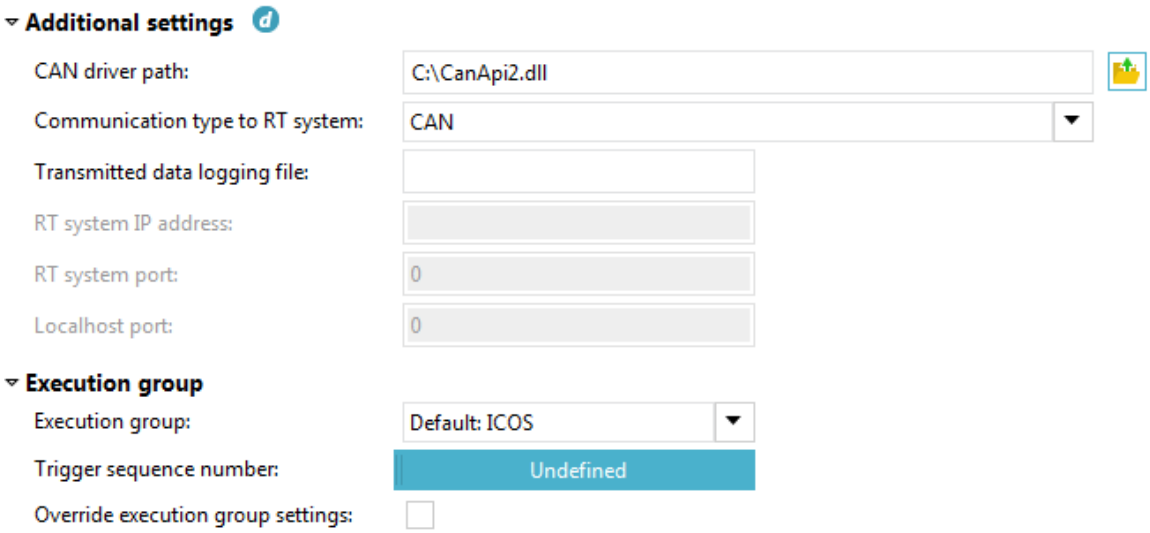

Abbildung 3.13: MC RT Block Parameter 2

Unter *Trigger sequence number* kann hier die Reihenfolge der Berechnung festgelegt werden.

Um ein Projekt mit einer CAN Verbindung zu testen, ist es nicht ausreichend, dass der PCAN USB Adapter an den PC angeschlossen ist, es muss auch eine aktive CAN Kommunikation existieren.

3.4.4 Embedded Mode

Es gibt die Möglichkeit, MC im sogenannten *Embedded Mode* zu betreiben. Das heißt, dass man die Simulation nicht in MC startet, sondern von einem externen Programm, hier AVLab.

In MC wird dafür der Block ausgewählt, welcher als Master dienen soll und anschließend der Button *ICOS embedded* in der oberen Menüleiste der Registerkarte *Home* ausgewählt (Abbildung 3.14 / links). Im nun erscheinenden Fenster wird das *Master element* und der *ICOS Port* ausgewählt (Abbildung 3.14 / rechts). Die voreingestellten Parameter sollten bei der oben genannten Vorgehensweise korrekt sein.

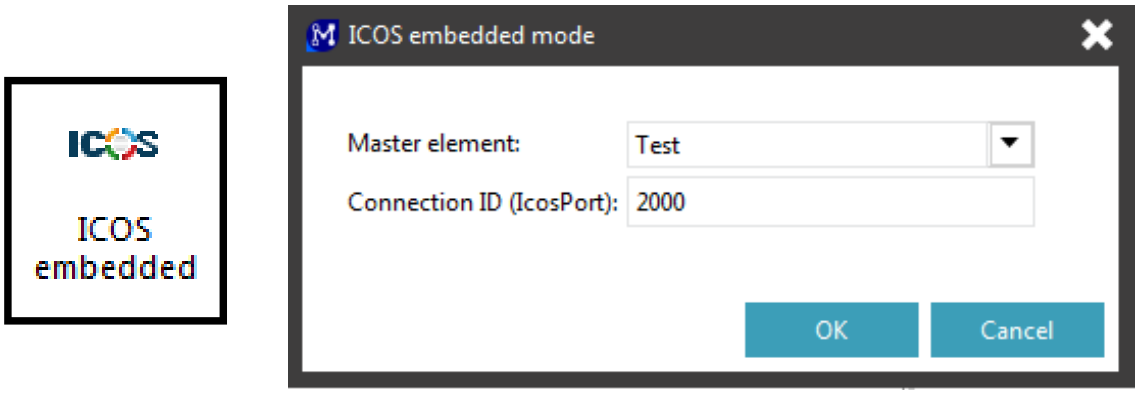

Abbildung 3.14: Embedded Mode aktivieren

Über das *Master element* wird die Simulation in MC dann gestartet und der *IcosPort* muss anschließend bei den *Execution settings* des Master-MATLAB-Blocks unter dem Parameter *MATLAB executable* eingetragen werden, wie schon in Abbildung 3.10 ersichtlich war.

Durch diese Aktion wird ein MATLAB Skript mit *.m Endung erstellt, in welcher der Projektname, das Installationsverzeichnis von MC und weitere Parameter enthalten sind. Der Inhalt dieser Datei darf nicht verändert werden, weil es andernfalls zu Simulationsproblemen kommt. Es wird in MC auch ein Job gestartet, der auf eine externe Aktivierung wartet. Man sieht dieses Verhalten im Statuswechsel des *Tasks* in der Registerkarte *Simulations* von *new* auf *ready* (Abbildung 3.15).

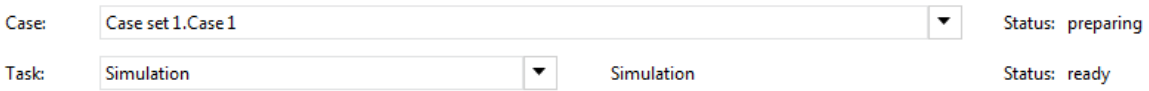

Abbildung 3.15: Start eines Tasks im Embedded Mode, Status: ready

Auch der Status des *Case* ändert sich von *new* auf *preparing* und der Button *Run* wird deaktiviert und *Stop* aktiviert.

Bleibt der Status auf *new*, so muss nur der *Job Server* (Abbildung 3.16) geöffnet werden und anschließend wieder geschlossen (v2016 ISO88)

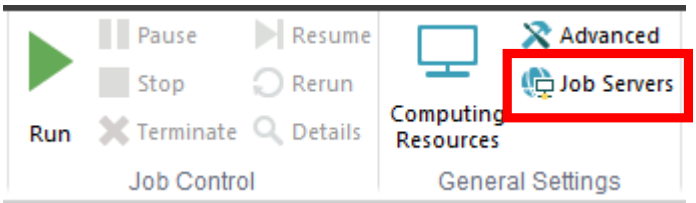

Abbildung 3.16: Job Server in der Registerkarte Simulations

Vor der ersten Simulation muss diese Prozedur durchlaufen werden, für weitere ist der Job bereits aktiv. Sollte die Simulation durch diverse Fehlermeldungen nicht starten, muss der Jobserver zurückgesetzt werden und MC beendet. Anschließend sind folgende Prozesse, soweit vorhanden, im Taskmanager zu schließen:

- *mysqld.exe*, danach sollte auch jms_mysqld.exe beendet sein
- *python.exe*, der Anwendungsprozess für MC
- *regsvr32.exe*, ein Microsoft© Registerserver, damit sich der Jobserver zurücksetzen lässt. Sonst zeigt MC „ongoing jobs“ an, welche aber nicht auffindbar sind.
  Dieser Prozess wird zusammen mit AVLab gestartet.
- *ICOS 2016 Remote Server*
- *ICOS 2016 runKernel*
- *icosm.exe, ICOS 2016*
- *mc_batch.exe*

In der Version v2015 ISO57 gab es noch einen weiteren Fehler, die Datei *submission1.py* wurde im falschen Ordner erstellt. Dieser Lapsus wurde jedoch behoben und somit wird hier auf den Workaround nicht näher eingegangen.

3.4.5 Berechnung auf mehrere Rechner verteilen

Für komplexe Simulationen mit vielen rechenaufwendigen Elementen besteht die Möglichkeit, einzelne Modelle auf verschiedenen Rechnern in einem Netzwerk zu verteilen um somit bei ausreichender Bandbreite einen Laufzeitvorteil zu erzielen.

Man muss dafür in MC unter den *Preferences* und *ICOS settings* einen *Remote server* erstellen (Abbildung 3.17). Bei der *IP address* ist die Host Netzwerkadresse oder der Computername einzutragen, der *Port* kann unverändert bleiben.

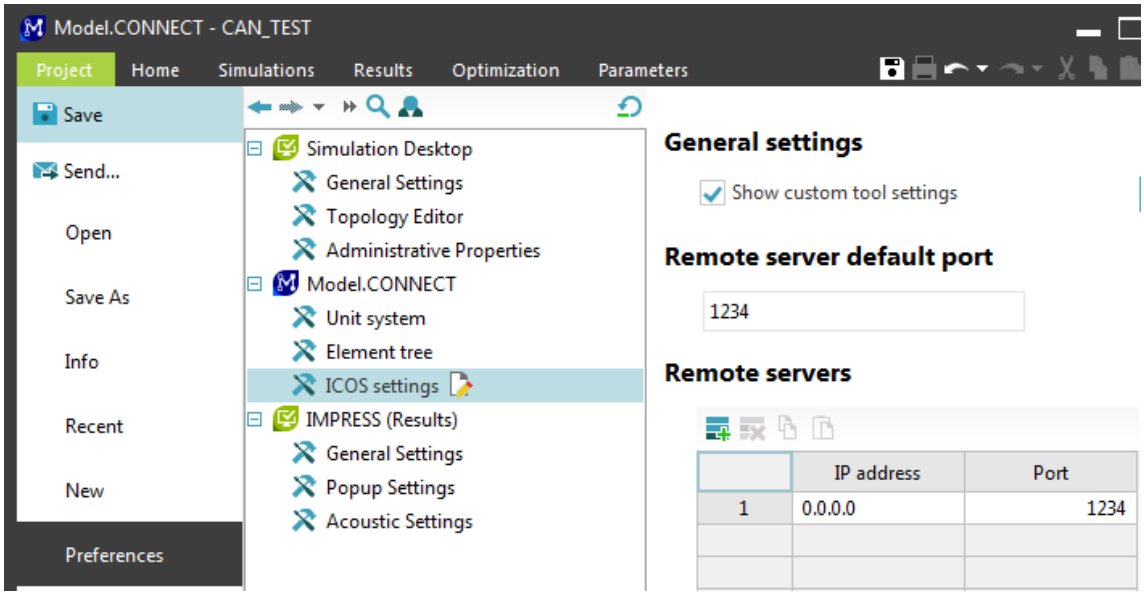

Abbildung 3.17: Einstellungen für verteiltes Rechnen von Modellen und den Additional Settings

Für das Modell, angenommen ein Simulink Modell, muss in MC ein Block eingefügt und die Parameter wie vorhin erläutert ausgefüllt werden.

Als erstes wird der Pfad des Modells auf dem anderen Rechner im Feld *Simulink model file* angegeben (Abbildung 3.9). Zusätzlich müssen in den *Execution Settings*, wie in Abbildung 3.10 ersichtlich, im Feld *Host* die IP oder der Rechnername des Hosts und der *Port* des *Remote Servers* angegeben werden.

Auf dem externen Arbeitsrechner muss nur der *ICOS remote server* mit dem gleichen Port gestartet werden. Dieser befindet sich im Installationsverzeichnis von MC und heißt in der Version 2016 ISO70 *remoteServer.exe*.

Diese Möglichkeit war für das Ziel dieser Arbeit unerheblich, da die Modelle praktisch verteilt gerechnet wurden, das Plant Modell auf einem PC und die HCU Software vom Steuergerät.

3.4.6 Verringerung der Synchronisationsdatenmenge

Üblicherweise werden bei der Anbindung von Simulink Modellen Sequenzen übergeben. Stellt man etwa in Simulink eine Schrittweite von 1ms (Mikroschritt) ein, und in MC eine Makroschrittweite von 10ms für dieses Modell, so werden zu jedem Makrozeitschritt zehn Werte übergeben. Wenn die restlichen Modelle, etwa Simulink- oder RT-Blöcke, aber ebenfalls mit einer Makroschrittweite von 10ms rechnen, dann reicht es lediglich den letzten Wert zu übergeben. Durch eine Verringerung der zu übertragenden Daten um den Faktor 10 kommt es zu einer merkbaren Verkürzung der Synchronisationszeit.

In MC kann man die Abtastzeit der Ausgangssignale aus den eingebundenen Modellen vorgeben. Dafür muss man bei den *Preferences* zu den *ICOS Settings* wechseln und die Option *Show custom tool settings* aktivieren, zu sehen im vorhergehenden Bild (Abbildung 3.17).

Durch diese Aktivierung hat sich in den Blöcken das zusätzliche Parameterfeld *Additional settings* geöffnet, in welchem man *Output Sample Time* und nach dem vorhin genannten Beispiel 0,01 (Sekunden) einträgt (Abbildung 3.18). Gäbe es ein Modell in dem MC Projekt, welches mit einer Makroschrittweite von 5ms arbeitet und Daten von diesem Block benötigt, müsste man einen Wert 0,005 wählen.

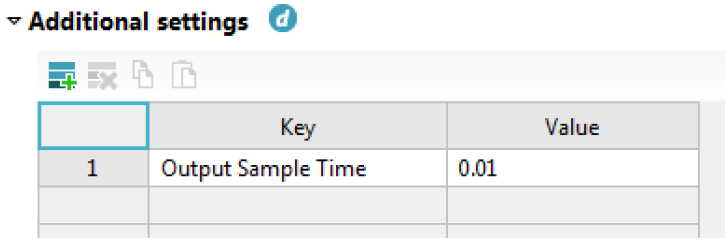

Abbildung 3.18: Aktivierung einer Abtastzeit für Ausgangssignale

Die Standardeinstellung ist hier *-1* (kontinuierlich), der Wert *null* bedeutet *inherited*. Damit würde der MATLAB Block die Abtastzeit vom Modell erben, diese Alternative ist jedoch nur mit Vorsicht zu genießen. Alle anderen Zahlen entsprechen einer Schrittweite in Sekunden. Diese Zeit darf natürlich nicht größer als die Makroschrittweite für diesen Block und nicht kleiner als die Mikroschrittweite für das Simulink Modell sein.

Der offensichtliche Nachteil dieser Einstellung ist die fehlende Datenmenge für die Extrapolation der Signale.

3.4.7 Programmhinweise

Nachfolgend werden weitere Fakten beschrieben, welche im Laufe der Programmentwicklung wichtig geworden sind und besondere Beachtung verdienen. Ein Hinweis gilt auch der Benennung der Projekte oder des Projektpfades, bei denen kein Umlaut enthalten sein darf.

DBC Byte-Ordnung

Für die CAN Kommunikation kann in der *.ini-Datei eine DBC Datei verlinkt werden, um den Aufwand zur Erstellung der ersteren zu reduzieren, die Byte-Ordnung sollte dann aus dem DBC gelesen werden. Es kann hier nur Intel verwendet werden, eine DBC Datei

mit Motorola[2] funktioniert in v2015 nicht und auch mit Ende dieser Arbeit war mit v2016 ISO88 die korrekte Übertragung von Signalen mit Motorola nicht möglich.

CAN Kanal

Wie schon im Kapitel 3.4.3 erläutert wurde, gibt es eine Zuordnung zwischen den CAN-USB Adaptern, *Handles* und dem CAN Kanal. Ab Version ISO88 v2016 wird, statt der einem *Handle* zugewiesenen ID, gleich die *Handle* Nummer in der *.ini- Datei eingetragen.

Ähnlichkeit von CAN Signalnamen

Es muss auch darauf geachtet werden, dass Signalnamen am CAN innerhalb einer Nachricht einander nicht beinhalten, da bei der Namenssuche das erste Ergebnis herangezogen wird. So wird das Signal *TorqueEM* mit dem Signal *Torque* verwechselt. Dieses Problem äußerte sich in einer Fehlermeldung bezüglich sich überlappender Signale.

Leerzeichen im CAN Signalnamen

Für die Erstellung der *.ini-Datei wurden die CAN Signalnamen aus einer Excel Tabelle kopiert. Bei ein paar Namen war, unbeabsichtigt, ein Leerzeichen als letztes Symbol eingefügt worden und wurde auch in die Konfiguration für den CAN übernommen. Da dieses Zeichen aber nicht im Signalnamen im DBC vorkam, wurde von MC ein unspezifischer Fehler ausgegeben. Durch Richtigstellung des Namens in der *.ini-Datei und anschließendem Aktualisieren des zugehörigen RT-Blocks wurde dieser Fehler sichtbar (Abbildung 3.19).

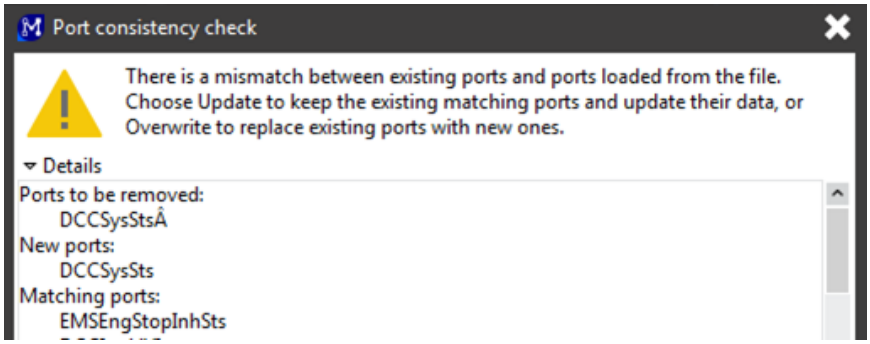

Abbildung 3.19: Fehlerbericht in MC aufgrund eines falschen Signalnamens

[2] Big-Endian

Das Signal *DCCSysSts* wurde in MC durch das Leerzeichen, welches als das Sonderzeichen *A-Dach* interpretiert wurde, nicht als gültig anerkannt.

Logsout Datei

Für den virtuellen SW Test, der genaue Aufbau wird im Kapitel 5 beschrieben, war die Speicherung der Signale in Simulink notwendig. Diese Signale wurden in einer Variable namens *logsout* gespeichert und konnten in eine *.mat Datei exportiert werden, um sie anschließend in einem anderen MATLAB Workspace zu laden.

Die Möglichkeit, diese Signale zu speichern gab es in v2016 nicht mehr und es wird seitens der Entwickler fieberhaft nach der Ursache gesucht.

CAN Buslast

Die Zykluszeiten von CAN Nachrichten sind in der DBC Datei angegeben. In MC werden jedoch alle Nachrichten mit der jeweiligen Makroschrittweite des RT Blocks gesendet. Beispielsweise wurden über den Private-CAN des aktuellen Projekts, neben zeitkritischen Signalen, die Zellspannungen der HV Batterie an die HVCU gesendet, um dort den SOC zu berechnen. Bei letzteren lagen die Zykluszeiten einer Nachricht mit vier Zellspannungen bei 100ms und eine mit vier SOCs sogar bei 1000ms. Durch die eingestellte Makroschrittweite des RT Blocks für diesen CAN wurden alle Nachrichten im 10ms Raster gesendet, was zu einer enorm hohen Buslast führt. Dieses Verhalten war aufgrund anderer Probleme mit dem CAN Bus noch nicht messbar, theoretisch führt es aber zu Schwierigkeiten bei der CAN Kommunikation.

3.5 PEAK Konfigurationsprogramme

Zur Simulation der Sensoren und Aktoren des Steuergeräts wurden Geräte von *PEAK Systems* verwendet, welche analoge sowie digitale Ein- und Ausgangssignale bereitstellen. Durch die Kombination mehrerer verschiedener Geräte, genannt *MicroMods*, konnten die gesamten Eingänge und Ausgänge des Steuergeräts abgedeckt werden. Im Einsatz waren zwei *MicroMods* mit analoger I/O und einer mit digitaler. Sie wurden über einen CAN Bus mit MC verbunden und auch über diesen konfiguriert. Mit Hilfe der mitgelieferten Software *PCAN-MicroMod Configuration* konnten die einzelnen Ein- und Ausgänge mit CAN Nachrichten und CAN Signalen verknüpft werden. Eine genaue Beschreibung und Konfigurationsanleitung findet man in den Handbüchern der Geräte.

4 Simulationskonfiguration

Die Software für das Hybridsteuergerät wird mit Hilfe von Simulink entwickelt. Sie besteht aus zahlreichen kleinen Modellen, welche für die verschiedensten Funktionen zuständig sind. So wird etwa der Drehmomentwunsch des Fahrers, ein redundant ausgeführtes Signal von einem E-Gas, ausgewertet, oder die Sicherheitsrelais für die Hochspannungsbatterie angesteuert.

Auch das Plant Modell, welches das Fahrzeug mit allen Eigenschaften und Steuergeräten widerspiegelt, ist in Simulink realisiert. Den Kern bildet ein AVL CRUISE Modell, in welchem ganze Fahrzeuge simuliert werden können, durch Zusammensetzen der einzelnen Fahrzeugkomponenten. Dazu gehören etwa die Art der Reifen, der Luftwiderstand, die Bremsleistung oder auch die Eigenschaften des Getriebes. Es sind alle Steuergeräte, wie die TCU oder BMS, im Plant Modell als einfache Funktionen ausgeführt.

4.1 Aufbau

Die Testanordnung ist in Abbildung 4.1 dargestellt. Da die HCU und das Plant Modell beide in Simulink programmiert sind, liegt ein einfacher Test über gegebene Schnittstellen nahe. Als Master diente die Softwareentwicklungsumgebung AVLab. Hier werden die Testfälle definiert und Stimuli Signale generiert, welche das Fahrzeugmodell anregen und von wo aus anschließend der Test startet. Alle relevanten auftretenden Signale, nicht nur die Kommunikation zwischen Plant und HCU, sondern auch HCU interne Signale, können erfasst werden, da sich beide Simulink Modelle im selben MATLAB Workspace befinden.

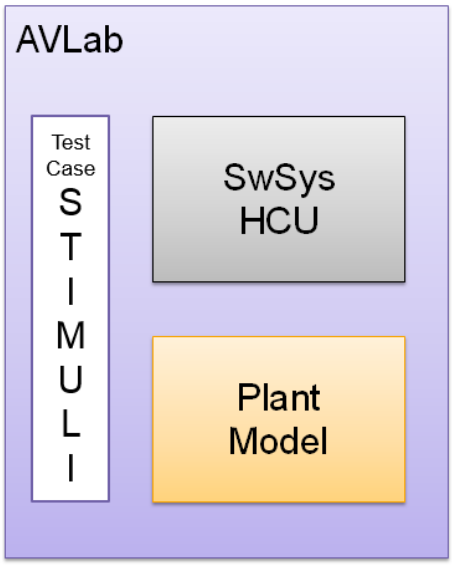

Abbildung 4.1: Ursprüngliche Simulationskonfiguration

Zusätzlich zu der bereits bekannten Struktur in Simulink (Abbildung 3.4) sind noch Speicherblöcke für jedes Signal vorhanden (Abbildung 4.2). Der Datenaustausch zwischen HCU und Plant findet über *DataStoreMemory* Blöcke statt. So schreibt das Plant das Signal *HVCUI_AcvGear*, den aktiven Gang, in den gleichnamigen *DataStoreWrite* Block, und die HCU holt sich diese Information über den entsprechenden *DataStoreRead* Block. In die andere Richtung funktioniert der Signalfluss genauso.

Der Datenaustausch über die Speicherblöcke ist sehr einfach und nützlich, bringt aber auch Probleme mit sich. Simulink ist hervorragend darin, algebraische Schleifen aufzuspüren. Diese Funktion wird mit einem Speicherblock zwischen Datenausgabe und Dateneingabe ausgehebelt, er entspricht keiner Zeitverzögerung.

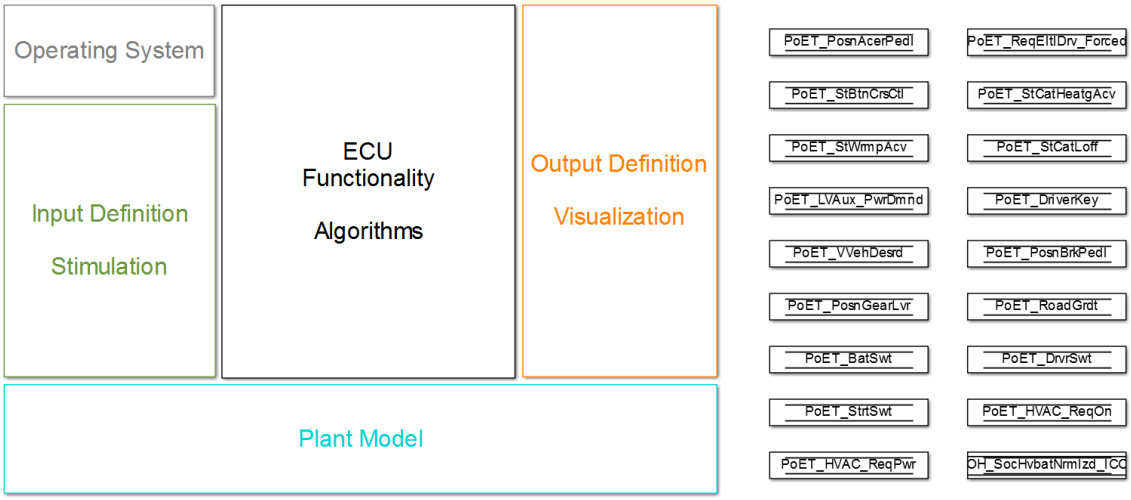

Abbildung 4.2: AVLab Standardstruktur mit Speicherblöcken

4.2 Testeigenschaften

Man bekommt mit diesem Test einen guten ersten Einblick der Funktionen und das Zusammenspiel der Einzelkomponenten und kann anhand von Diagrammen abschätzen, ob die gewünschten Funktionen richtig implementiert worden sind.

Der Computer übernimmt die gesamte Rechenleistung, sowohl von HCU als auch vom Plant. Verwendet man zur Simulation einen gut ausgerüsteten Computer, so ist diese schneller und man kann weder über die Rechenzeit im Steuergerät, noch auf die Echtzeitfähigkeit des Plant Modells, welche später zu Tragen kommt, eine Aussage treffen.

Die CAN Signale, welche zwischen HCU und Plant ausgetauscht werden sind hier sofort verfügbar, was die Realität in einem Fahrzeug aber nicht widerspiegelt. Es gibt tatsächlich Verzögerungen in der Signalübertragung aufgrund von parasitären Kapazitäten und der endlichen Geschwindigkeit der CAN Busse. Daher tritt eine unerwünschte Totzeit zwischen Regler und Strecke auf, welche in dieser Anordnung vernachlässigt wird.

4.3 Testfall 12

Für die weitere Validierung des Tests in unterschiedlichen Konfigurationen wurde ein spezieller Testverlauf verwendet. Die gewonnenen Ergebnisse wurden später mit dem virtuellen und realen SW Test als Vergleich herangezogen.

Der Testfall 12 beinhaltet zwei Beschleunigungsphasen, zwei Verzögerungsphasen und endet mit einem Stopp des Fahrzeugs. Die Signale eines simulierten Fahrers, wie die Position des Fahrpedals oder des Bremspedals liegen bereits vor.

5 Virtueller Software Test

Der originale Test, der nur in Simulink implementiert war, wurde im nächsten Schritt erweitert um das Co-Simulationsprogramm Model.CONNECT. Der Austausch der Daten zwischen der HCU und dem Plant Modell fand nun nicht mehr in Simulink statt, sondern über MC. Diese Testanordnung diente nur zur Validierung der Testdaten und war auch dem realen Verhalten im Fahrzeug bezüglich Laufzeiten ähnlicher.

Im realen Steuergerät, der HVCU, nicht zu verwechseln mit der Hybridsteuergerätsoftware HCU, befand sich außer der Hybridsteuerung noch das BMS und die Steuerung der C0 Kupplung, welche sich bei einem P2-Hybridfahrzeug zwischen dem Verbrennungsmotor und dem Elektromotor befindet. Für die Signalaufbereitung gibt es noch die BSW, welcher die Treiber für die Hardware zur Verfügung stellt und den CIL, den man als Middleware ansehen kann. Diese Architektur wurde gewählt, da das C0 nur geringen Rechenaufwand benötigt, aber dafür viele Hardwareschnittstellen, im Gegensatz dazu erfordert das BMS und die HCU SW eine hohe Rechenleistung.

In Abbildung 5.1 wird die HVCU mit ihren SW Modulen gezeigt. Die C0 und BMS werden im Plant simuliert, somit müssen nur die tatsächlich von der HCU verwendeten Signale bereitgestellt werden. Diese entsprechen jenen zwischen HCU und CIL.

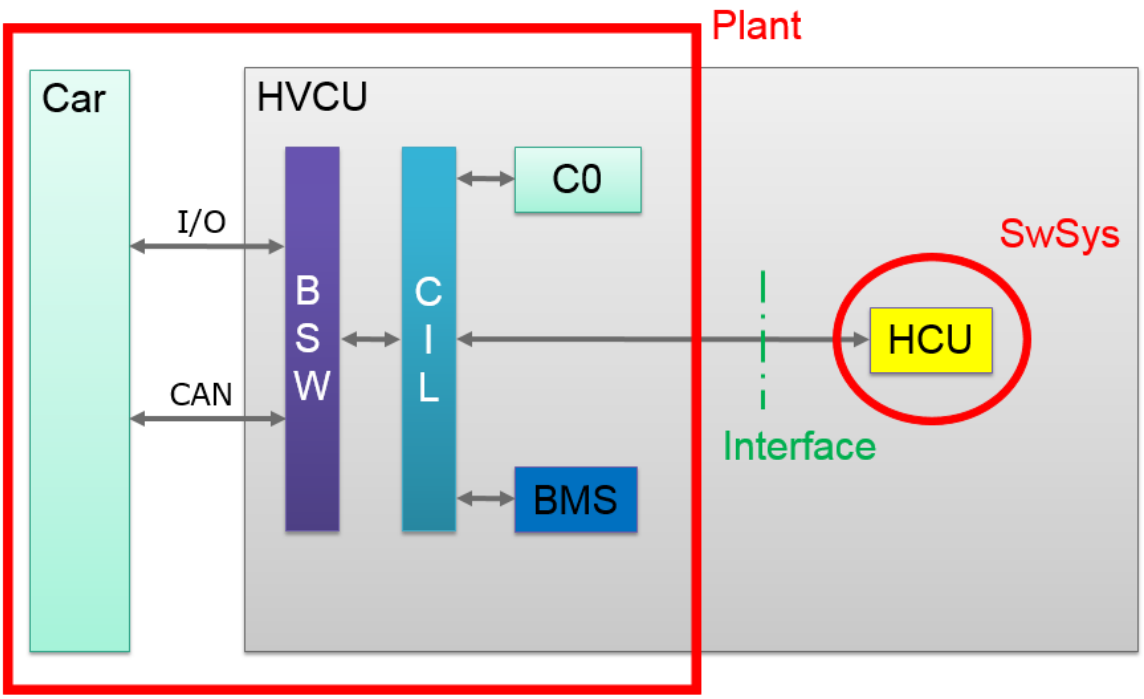

Abbildung 5.1: Testaufbau für den virtuellen Test

Der Datenaustausch zwischen C0 und HCU, etwa der Status der Kupplung oder eine Drehmomentanforderung, findet über die Schnittstelle und somit über MC statt. Auch zwischen BMS und der HCU gibt es einen Informationsaustausch über den SOC der HV Batterie und weitere Parameter. Diese Interaktionen, welche im Fahrzeug im realen Steuergerät intern ablaufen und dieses nicht verlassen, müssen hier berücksichtigt werden.

5.1 Testanordnung

Es wurde das HCU Modell aus der AVLab Umgebung entnommen, damit die Kommunikation mit dem Plant ausschließlich über MC stattfindet (Abbildung 5.2).

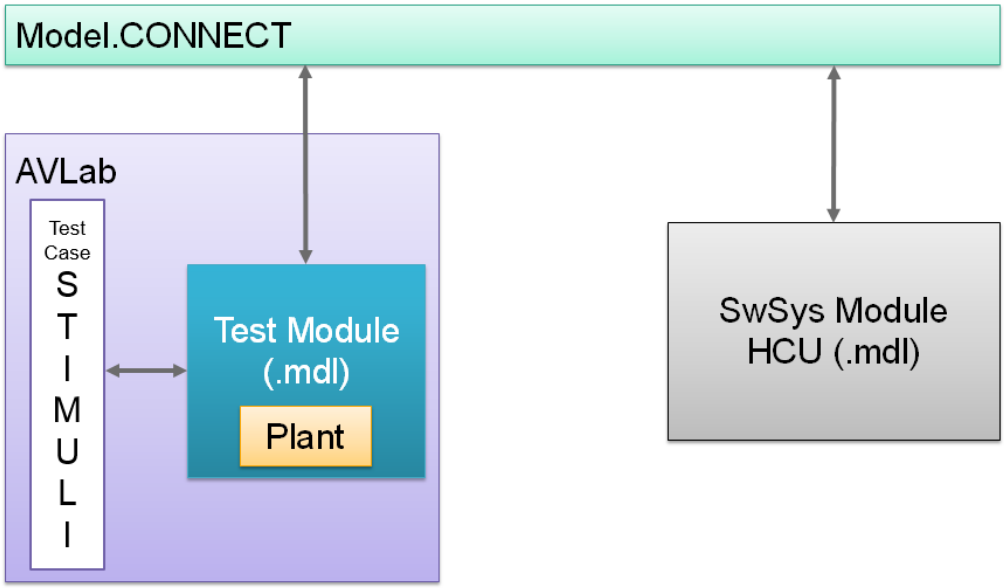

Abbildung 5.2: MC als Schnittstelle zwischen den Modellen

Beide Modelle wurden über die MATLAB Schnittstelle in MC verbunden. Durch Extraktion der HCU SW aus dem Test Module, in dem sich auch das Plant Modell befindet, kann man beide Modelle parallel rechnen um die Simulationszeit zu verkürzen.

5.2 Änderungen der Simulink Modelle

Um eine Schnittstelle zwischen MC und den Modellen zu schaffen, mussten letztere geringfügig verändert und erweitert werden. Es wurden neben den *ICOS Blöcken* auch Befehle zur Speicherung und Übertragung der Simulationsergebnisse zu AVLab benötigt.

5.2.1 HCU

Da die Software einer ständigen Weiterentwicklung unterlag und auch die Austauschbarkeit für weitere Projekte einfach gegeben sein sollte, durfte dieses Modell nicht verändert werden. Die Konstruktion der Schnittstelle erfolge durch einen „Rahmen“ übers Modell.

5.2.2 Rahmen

Dieser Rahmen übernahm die Kommunikation mit MC und reichte die Signale an die, als *Referenced model* eingebundene, *SwSys* weiter (Abbildung 5.3). Über die Farben der Blöcke konnte man einfach auf die Zugehörigkeit der einzelnen Signale auf einen CAN Bus schließen.

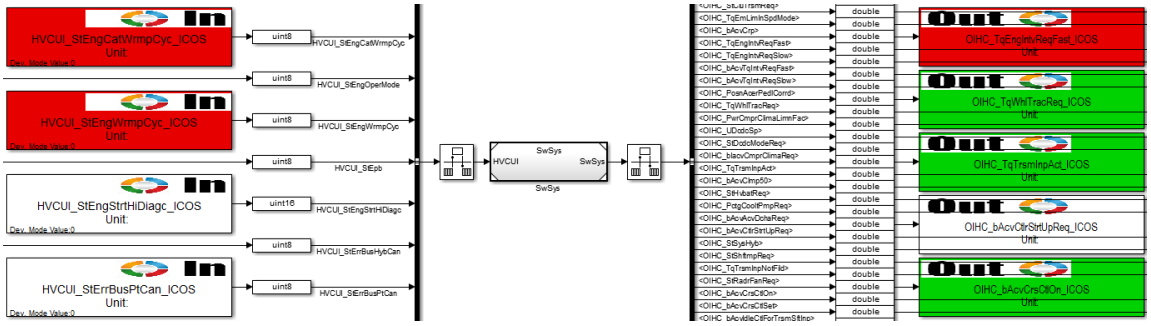

Abbildung 5.3: Teil des Rahmens um ursprüngliches HCU Modell

Es wurden *ICOS Out* und *ICOS In* Blöcke in ein Simulink Modell eingefügt und die Signale zu einem Bus zusammengefasst. Dazwischen befand sich der Cast der Datentypen auf den im Bus definierten Typ. Bei den Eigenschaften des *BusCreators* musste die Funktion *Output as nonvirtual bus* angewendet werden, da die *SwSys* als *Referenced Model* eingebunden wurde und dieser Block keine virtuellen Busse erlaubt.

Der Hauptunterschied dieser Arten von Bussen liegt im Zugriff auf die Daten. Ist ein Block an einen virtuellen Bus angeschlossen, wird er seine Eingänge lesen und Ausgänge schreiben, in dem er auf den allozierten Speicher der Komponentensignale zugreift. Diese Speicherplätze liegen nicht notwendigerweise nebeneinander und es existiert kein weiterer Zwischenspeicher. Verbindet man einen Block mit einem nicht-virtuellen Bus, greift seine I/O nur auf Kopien der Komponentensignale zu. Diese Kopien sind in angrenzenden Speicherplätzen und werden vom Bus reserviert. Es ergibt sich hier ein erhöhter Bedarf an freiem Speicher und die Simulation wird langsamer, da Kopien von allen Signalen erstellt werden müssen. Der Vorteil liegt hier in der Anordnung

der Elemente in einer Struktur, was hilfreich bei der Nachverfolgung der Analogie zwischen Modell und Code sein kann [23].

Zwischen den Ein- und Ausgangssignalen und dem Modell war noch eine *RateTransition* notwendig, um einen konsistenten Datentransfer bei Systemen unterschiedlicher Zykluszeiten zu gewährleisten. Bei den Ausgangssignalen verhielt es sich ähnlich wie bei den Eingangssignalen, hier mussten alle Signal-Datentypen auf *double* konvertiert werden.

5.2.3 AVLab Schnittstellenmodell

Die größeren Änderungen wurden im Modell *SwSys_Test* gemacht. Statt die HCU wie bisher als *Referenced Model* in die unterste Ebene des Blocks *ECU Functionality* im *SwSys_Test* einzubinden, wurde hier die Schnittstelle zu MC benötigt.

Der erste Versuch, nun zwischen den beiden Modellen MC einzubinden, war sehr naheliegend. Statt auf das HCU Modell zu referenzieren, wurden *ICOS Out* und *ICOS In* Blöcke eingefügt, welche die Signale zu den entsprechenden Gegenstücken im Rahmen schicken sollten. Etwa hatte das Ausgangssignal für die Temperatur des Getriebeöls ein gleichnamiges Eingangssignal im Frame um die HCU.

Diese Anordnung hat aber nicht funktioniert, da das Co-Simulation-Programm MC als Master zwischen den eingebundenen Modellen agiert. In einem einstellbaren Zyklus, Makroschritten, werden Daten zwischen den einzelnen Simulationen ausgetauscht. Es ist nicht zulässig, die Simulationsdaten eines Modells zu dessen Bedingungen preiszugeben. Diese Funktion erfüllte aber das *function call subsystem*, welches sich Ebenen über der *ICOS* Schnittstelle befand und stellte so eine Verletzung der Simulationsregeln für MC dar.

Als beste Lösung für dieses Problem wurde der Datenaustausch auf die höchste Ebene verlagert. Statt den *ICOS Blöcken* in der untersten Ebene wurden diese Signale in weiteren *DataStore* Blöcken gespeichert und die oberste um das selbst erstellte Subsystem *ICOS_to_System* ergänzt (Abbildung 5.4).

Zusätzlich kann man in diesem Bild einen Block für den *TargetLink Main Dialog* und den *MiL Handler* erkennen. In ersterem gibt es zahlreiche Einstellungsmöglichkeiten etwa für die Codegenerierung und Simulation.

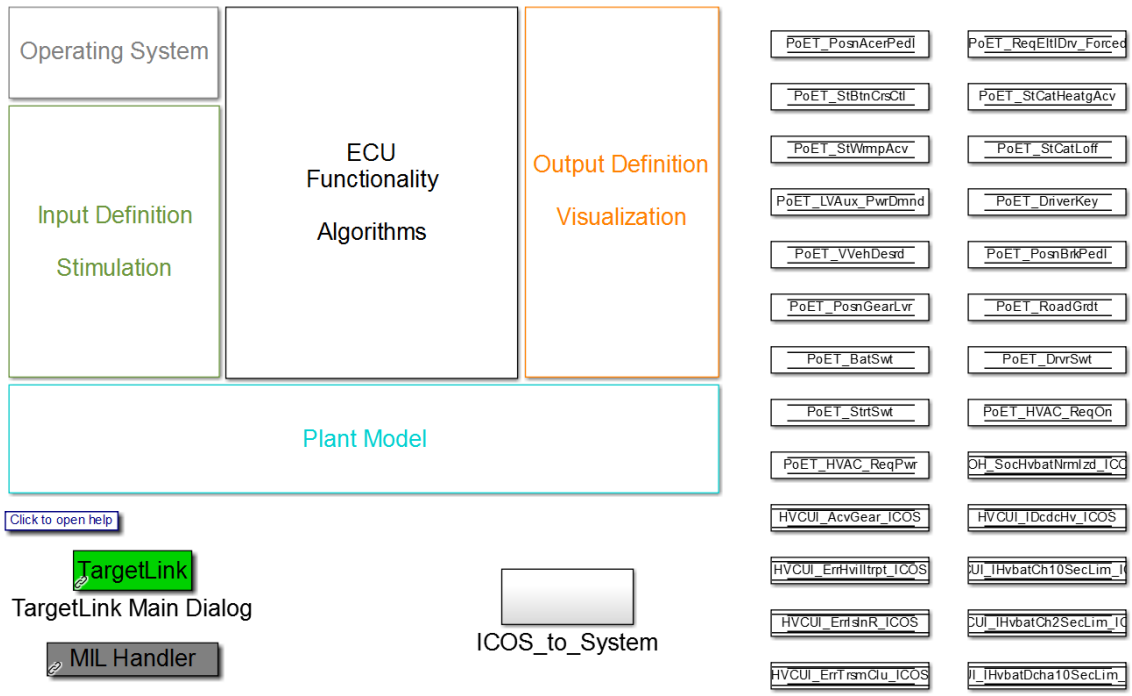

Abbildung 5.4: Einfügen eines Subsystems, um die ICOS Schnittstelle zykluszeitunabhängig zu machen

In dem Subsystem wurden nun die neuen *DataStore* Blöcke ausgelesen und über die *ICOS* Schnittstelle an MC gesendet, zugleich Daten aus MC empfangen und auf *DataStoreWrite* Bausteine geschrieben (Abbildung 5.5). Somit wurden die Signale zyklisch vom virtuellen Betriebssystem des Steuergeräts aktualisiert und konnten zu beliebigen Zeiten von MC abgefragt werden.

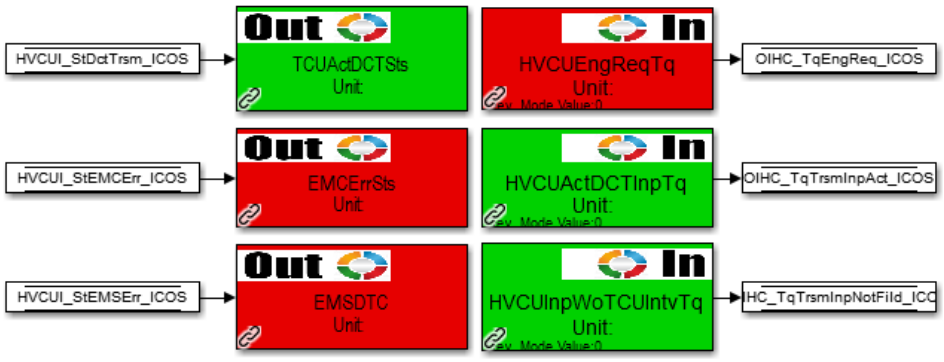

Abbildung 5.5: DataStore Blöcke schreiben auf und lesen von ICOS Blöcken

Der Nachteil dieses Aufbaus lag in der Verdoppelung der Speicherblöcke, da es nun für jedes Signal im *function call subsystem* ein weiteres zu der *ICOS* Schnittstelle gab.

Es wurden die ursprünglichen Speicherblöcke kopiert und an die neuen Signalnamen die Endung „_ICOS“ angefügt. Für jedes Signal im Plant Modell war in ADD ein *Simulink.Signal.Objekt* definiert, in welchem verschiedene Parameter wie Grenzwerte, Datentyp Beschreibung oder Einheit hinterlegt waren. Lud man eine Kalibrationsdatei für einen Testfall, so konnte für diese Signale ein Initialisierungswert vorgegeben werden. Diese Eigenschaften gab es für die neuen Signale nicht, somit durfte bei den *DataStoreMemory* Blöcken bei Code Generation die Eigenschaft *Data store name must resolve to Simulink signal object* nicht ausgewählt werden und ebenso musste ein *Initial value* angegeben werden. Diesen Ausgangswert konnte man aus der DBC Datei auslesen. Um ein über- oder unterschreiten der Grenzwerte zu vermeiden ist es nicht ratsam alle Werte auf *null* zu setzen.

5.3 Zusätzliche Funktionen in Simulink

Damit der Test fehlerfrei ablaufen konnte, musste eine Parametrierung der Modelle durchgeführt werden. Ein Nachteil in dieser Testanordnung liegt in der Verfügbarkeit der Signale. Während beim originalen Test beide Simulink Modelle in einer MATLAB-Instanz geöffnet waren und alle auftretenden Signale in AVLab aufgezeichnet wurden, so werden nun zwei MATLAB Instanzen geöffnet. AVLab hat hier nur Zugriff auf die Signale von jener, welche es selbst gestartet hat, das Plant und der Kommunikation zur HCU. Dieser Umstand wurde mit den AVLab Entwicklern besprochen, aber bis zum Ende der Masterarbeit lagen noch keine integrierten Lösungen vor. Mit einem einfachen Workaround, beschrieben in Kapitel 5.3.1, konnte dieses Problem vorerst gelöst werden.

5.3.1 SwSys_Test

Im Modell in dem sich auch das Plant befand, im Projekt *SwSys_Test* genannt, waren bei den *Callbacks* Skripts und Befehle einzugeben, welche zu gegebener Zeit ausgeführt wurden. In der *InitFcn* mussten zuerst die Systemvariablen und Busse in den MATLAB Workspace geladen werden. Anschließend wurden diese Variablen zusammen mit der von AVLab geladenen, testfallspezifischen Konfiguration in eine *.mat Datei gespeichert. In dieser Funktion musste nun auch das für den *Embedded Mode* erstellte Skript, hier *Test* genannt, aufgerufen werden (Abbildung 5.6).

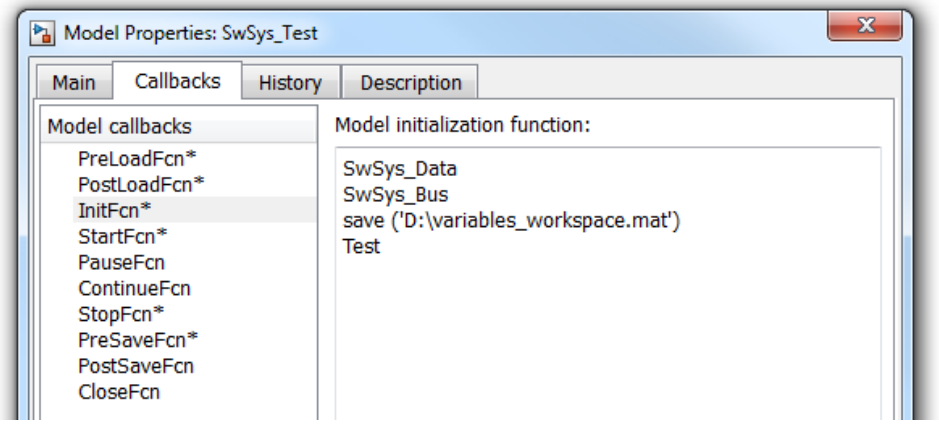

Abbildung 5.6: Callbacks für SwSys_Test als InitFcn

In der *StopFcn* (Abbildung 5.7) wurde eine von der HCU erstellte *.mat-Datei geladen. In dieser befanden sich die HCU internen Signale, welche für den Vergleich der Simulationsdaten benötigt wurden.

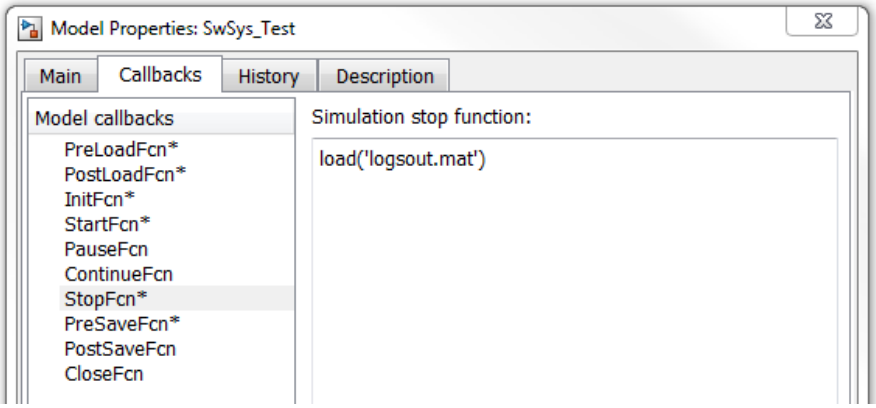

Abbildung 5.7: Callbacks für SwSys_Test als StopFcn

5.3.2 SwSys

Um alle in der HCU auftretenden Signale zu erfassen, musste im Modell *SwSys* unter *Model Configuration Parameters* in der Registerkarte *Data Import/Export* das *Signal logging* aktiviert werden (Abbildung 5.8).

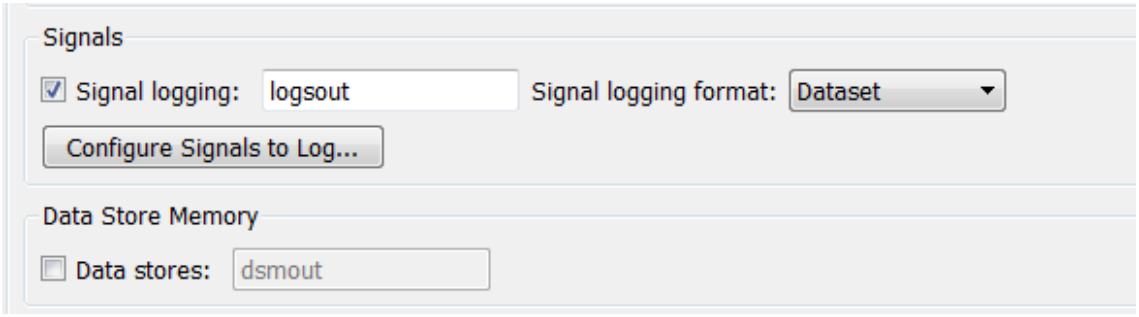

Abbildung 5.8: Aktivierung von Signal logging in der HCU

In diesem Modell mussten keine Callback Funktionen ausgeführt werden.

5.3.3 SwSys_frame

Bei der *PreLoadFcn* im Rahmen, welche noch vor dem Öffnen des Simulink Modells bearbeitet wird, wurde als erstes der Projektpfad in MATLAB eingebunden. Anschließend mussten noch die Variablen aus dem anderen Workspace geladen werden (Abbildung 5.9).

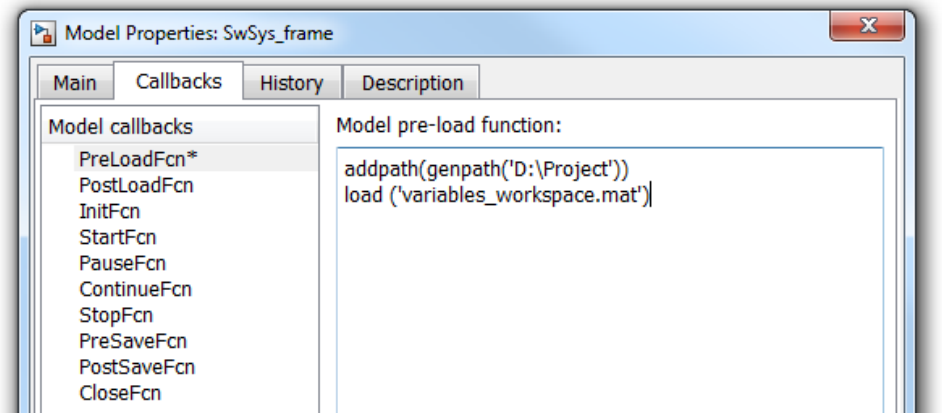

Abbildung 5.9: Callbacks für SwSys_frame als PreLoadFcn

Nach Abschluss der Simulation waren nun die mitgeloggten Signale in einer *.mat Datei zu speichern, die anschließend von AVLab zur Auswertung der Daten verwendet werden konnte (Abbildung 5.10).

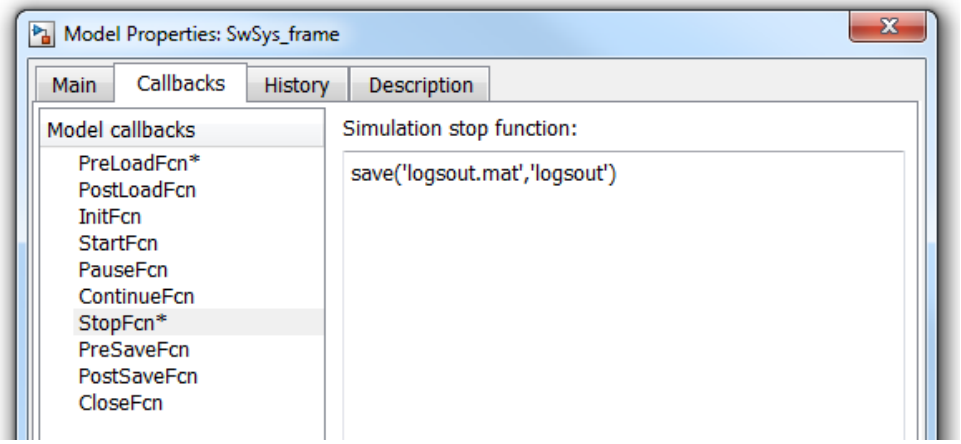

Abbildung 5.10: Callbacks für SwSys_frame als StopFcn

5.4 Aufbau in MC

Als erster Versuch wurden drei Simulink Modelle erstellt und über die MATLAB *ICOS Tool Interfaces* in MC eingebunden. In einem war das Plant, in einem anderen die Schnittstelle zu AVLab (*Test*) und im dritten die HCU Software (*Sys*). Mit dieser Anordnung, wie in Abbildung 5.11 dargestellt, wäre es einfach möglich, das Fahrzeugmodell oder die Systemsoftware auszutauschen. In dem mittleren Block war AVLab eingebunden, um alle Signale zwischen den Modellen aufzuzeichnen.

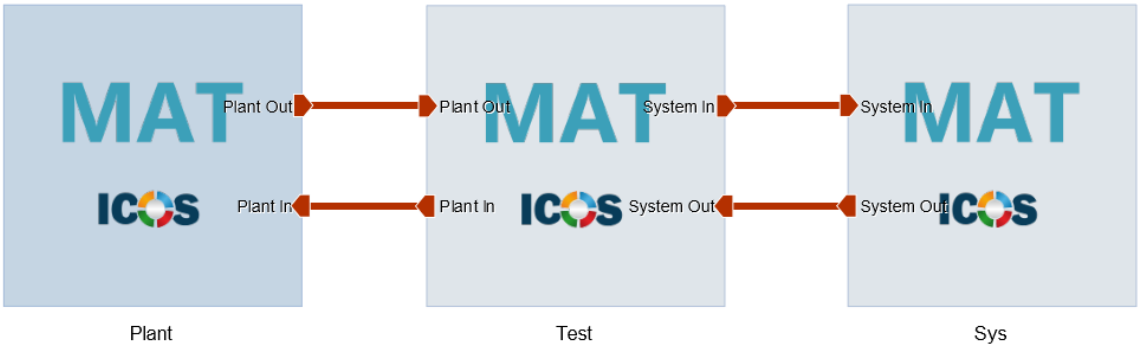

Abbildung 5.11: Erste Modellanordnung in MC

Für Versuche wurden statt den umfangreichen Modellen einfache Signale zwischen den Elementen ausgetauscht. Die HCU schickte einen konstanten Wert und eine Summe ans Plant. Die Konstante wurde zur Summe addiert und wieder zurück ans System geschickt (Abbildung 5.12). Zusätzlich wurde noch ein Zeitstempel übertragen. Im Modell Test wurden die Signale nur durchgeschleift und aufgezeichnet. Jeder Block in MC hatte eine Makroschrittweite von 10ms.

Bei einer Simulationszeit von 10 Sekunden sollte es zu einer Gesamtsumme von 1000 kommen. Jedoch ergab sich bei einer parallelen Berechnung ein Wert von 500.

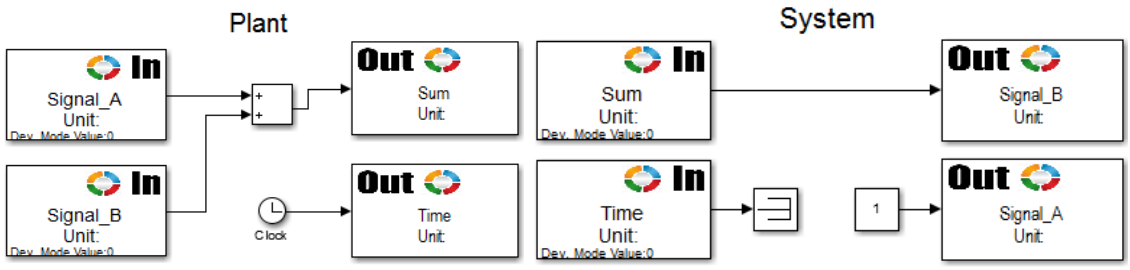

Abbildung 5.12: Einfaches Testmodell

Mit der Einstellung *sequential* und der Reihenfolge von links nach rechts wie in Abbildung 5.11 kam ebenfalls der Wert 500 heraus. Es stellte sich heraus, dass mit jeder beliebigen Berechnungsabfolge immer die Hälfte des erwarteten Endwerts vorlag, da nur bei jedem zweiten Simulationsschritt die aktuellsten Ergebnisse weiterverarbeitet werden konnten.

Nimmt man als Beispiel die Reihung Plant, dann Test und anschließend System. In dieser Serie werden die gegenwärtigen Daten aus dem Plant Modell korrekt an die HCU über das AVLab Modell weitergegeben, und am Ende dieses Makroschritts stehen deren Ergebnisse an den Ausgängen des HCU Modells bereit. Anstatt nun die Resultate zurück an das Fahrzeugmodell zu übermitteln, wird jedoch wieder in der gleichen Reihenfolge begonnen. Das Plant hat jetzt noch die veralteten Daten an den Eingängen, berechnet ein falsches Ergebnis und legt diese an seine Ausgänge. Wenn nun das mittlere Modell seine Eingänge aktualisiert, bekommt es die Daten aus dem letzten Rechenschritt vom System und legt diese auf seine Ausgänge zum Plant. Auch die HCU bekommt die überholten Daten und berechnet ein inkorrektes Ergebnis. Wenn nun der dritte Makroschritt beginnt, liest das Fahrzeugmodell endlich die neuen, aber bereits veralteten Ergebnisse aus der HCU ein und verarbeitet sie. Durch den Verlust eines Zeitschritts ist diese Topologie mit drei Blöcken für diese Anwendung nicht möglich.

Um nur zwei Simulink Modelle zu erhalten wurde das Plant Modell, wie ursprünglich, in das Modell mit der AVLab Schnittstelle eingebaut (Abbildung 5.13).

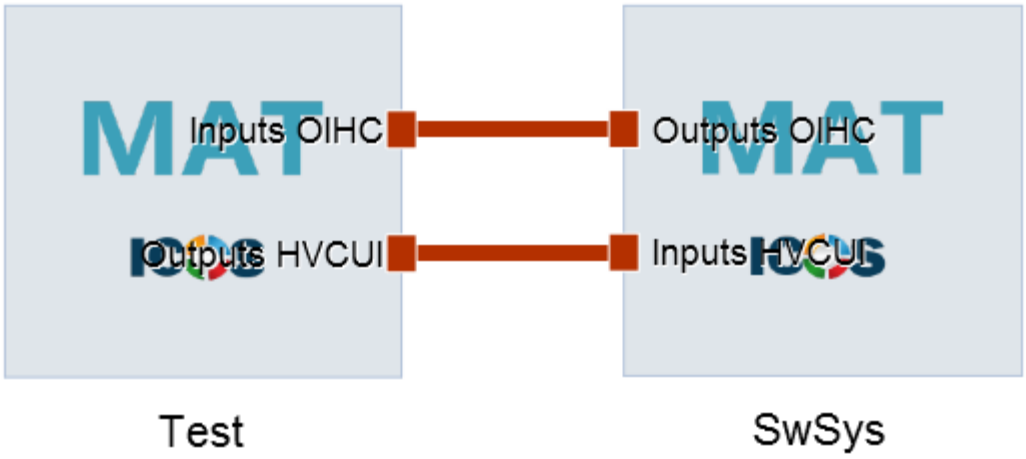

Abbildung 5.13: Aufbau in MC

Die Modelle des Plants, hier Test genannt, und der *SwSys*, wurden über zwei MATLAB Blöcke eingebunden. Letzteres bestand aus dem Rahmen mit referenzierter HCU SW. Durch diese Verlinkung der Modelle wurden alle *ICOS* Ein- und Ausgänge der Modelle in MC sichtbar. Da es sich um über 100 Signale handelte, wurden sie in MC zur besseren Übersicht in sogenannte *Bundles* gepackt. Die Schrittweite wurde gleich der Zykluszeit der HCU in Simulink in beiden Blöcken auf 10ms gesetzt.

5.5 Testablauf

Als erstes musste sichergestellt werden, dass in MC der Status des Tasks auf *ready* stand und ein aktiver Job auf Ausführung wartete, sowie eine *TargetLink* und *Stateflow* Lizenz verfügbar waren und natürlich auch eine für MATLAB und Simulink.

Startete man nun die Simulation in AVLab, wurden als erstes alle notwendigen Variablen in den MATLAB Workspace geladen und anschließend das *SwSys_miltest* geöffnet, welches das Plant enthält. Automatisch wurden die Stimuli Signale an vorgefertigten Schnittstellen angefügt und das Modell gestartet. Da nun als Initialisierung ein MATLAB Skript vorlag, wurde dieses ausgeführt und in MC die nächsten Schritte durchgeführt.

MC startete bei diesem Aufbau eine weitere MATLAB Instanz und führte das *SwSys* Modell aus. Dieses lud die Variablen aus dem ersten Workspace und startete mit der Simulation. Anschließend wurden die Simulationssignale über MC ausgetauscht und man konnte in MC unter der Registerkarte *Simulations* beliebige Signale, etwa mit einem *Curve Monitor*, beobachten.

5.6 Ergebnisse

Die Verifikation der Simulationsergebnisse erfolgte durch Vergleich mit jenen aus dem ursprünglichen Test. Dafür wurden die Daten beider in eine Excel Datei geladen und Diagramme erstellt, welche jeweils das gleiche Signal der beiden Simulationen darstellten, sowie Diagramme der Differenz der Signale.

Für den Vergleich wird nun auf zwei Signale näher eingegangen, welche alle aufgetretenen Simulationsbesonderheiten widerspiegeln. Bis auf zwei Signale war bei allen der Signalverlauf identisch. Im nachfolgen Diagramm 1 ist die Drehmomentanforderung der HVCU an den Verbrennungsmotor und Elektromotor dargestellt. Die orange Kurve stellt den Signalverlauf bei der ursprünglichen MiL Simulation dar, die blaue die Ergebnisse der Simulation über MC.

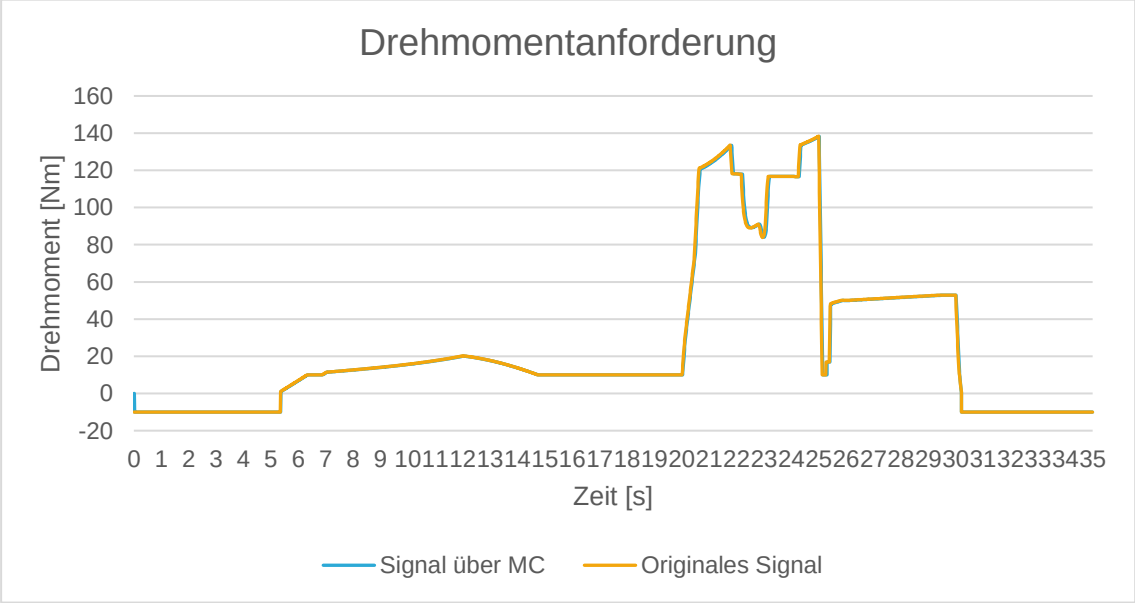

Diagramm 1: Vergleich der Simulationsergebnisse anhand der Drehmomentanforderung

Zu Beginn und Ende der Simulation wird ein negatives Moment angefordert, welches durch Aktivierung der Bremse oder Rekuperation erreicht werden kann, dazwischen wird mit konstanter Geschwindigkeit gefahren und beschleunigt. Bei den ersten Werten der Simulation wird sichtbar, dass der Initialisierungswert für dieses Signal in Simulink auf null gesetzt wurde, da es nicht kalibriert wird. Dieser Fehler wird jedoch beim ersten Simulationsschritt mit neuen Daten korrigiert.

Bei näherer Betrachtung der Kurven fiel auf, dass es bei allen Signalen zu einer Zeitverzögerung von etwa 20ms kommt. In Diagramm 2 wird ein Ausschnitt von 20s bis 26s dargestellt und hier wird diese Verschiebung deutlich.

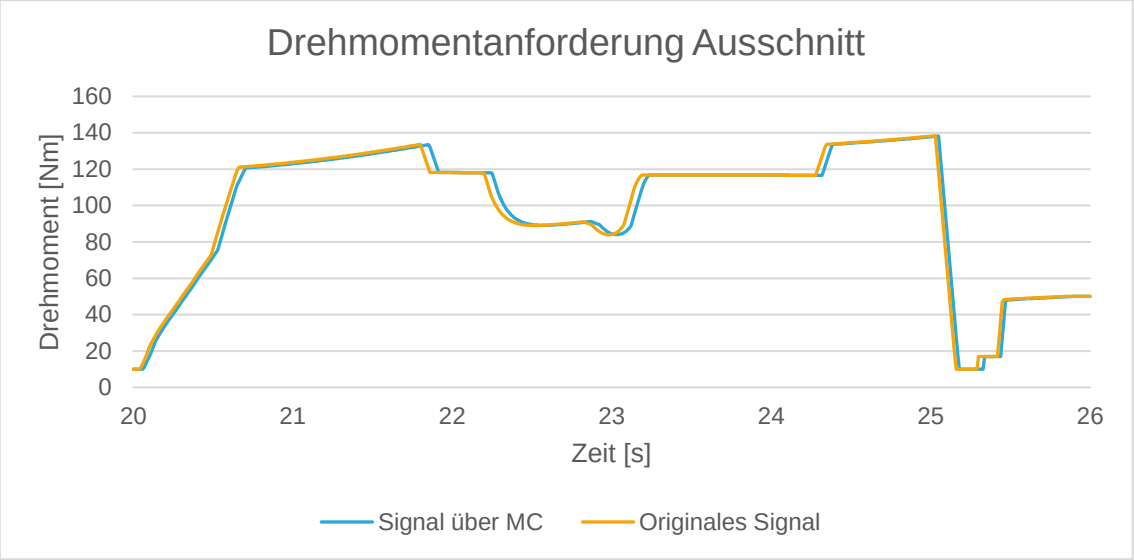

Diagramm 2: Vergleich der Simulationsergebnisse anhand der Drehmomentanforderung im Detail

Dieses Verhalten ist auf die Zeitverzögerung der Berechnung von Signalen aufgrund von Latenzzeiten der Übertragung zurückzuführen. Möchte man die Ergebnisse automatisch mit AVLab vergleichen, muss man die Resultate aus der Simulation über MC um zwei Zeitschritte nach hinten verschieben, da sonst immer ein Fehler in der Gegenüberstellung auftreten wird.

Eines der beiden Signale, welches nicht nur eine zeitliche Abweichung zeigte, ist in Diagramm 3 abgebildet, der Strom durch den Gleichspannungswandler auf der Hochspannungsseite.

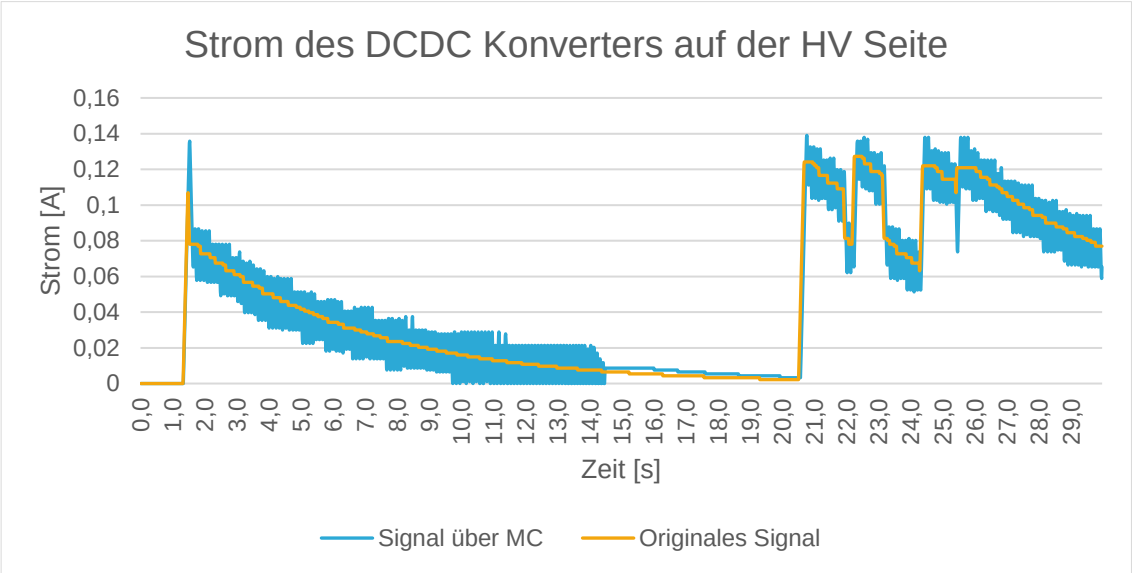

Diagramm 3: Vergleich der Simulationsergebnisse anhand des DCDC Stroms

Der Strom weist starke Schwingungen auf, welche vermutlich aufgrund von Totzeiten in der Regelung auftreten. Es wurden verschiedene Möglichkeiten ausprobiert, um das Simulationsergebnis zu verbessen. Ursprünglich war bei beiden Modelle in MC, das Plant und HCU, eine Makroschrittweite von 10ms eingestellt und alle Signale wurden mit der Option ZOH interpoliert. Zuerst wurde die Interpolationsmethode des Signals geändert. Bei FOH zeigte sich keinerlei Änderung, und bei SOH wich der Verlauf noch stärker ab, die Schwingungen wurden mehr.

Abhilfe konnte erst eine Reduzierung der Makroschrittweite des Plant Modells schaffen. Dabei machte es keinen Unterschied im Ergebnis, ob man 9ms oder 1ms auswählte, durchaus jedoch in der Berechnungszeit. Durch die höhere Berechnungsfrequenz des Plant Modells, hatte die HCU immer aktuelle Daten zur Verfügung und lieferte eine passende Stellgröße.

Ein Nachteil dieser Lösung war der Verlust von Ergebnissen, da in jedem zehnten Zeitschritt das Plant zwei Mal berechnet wurde, bevor die HCU die Daten abfragen konnte. Dies ergab sich aus dem Ausführungsalgorithmus in MC, welcher immer das nächste ausstehende Modell synchronisierte.

6 Realer Software Test

Im letzten Schritt wurde die HCU als Simulink Modell gegen ein reales Steuergerät, der HVCU, getauscht. Die Signale aus dem Plant wurden somit in MC auf CAN Busse gelegt und, wie auch im Fahrzeug, ans Steuergerät gesendet. Im Gegensatz zum virtuellen SW Test verschob sich hier die Schnittstelle (Abbildung 6.1).

Die C0 und BMS waren im Plant enthalten und befanden sich nun in der HVCU. Das BMS benötigte die einzelnen Zellspannungen der HV Batterie und berechnete den SOC für die einzelnen Zellen, welche sie über CAN an andere Steuergeräte schickte. Diese Funktion wurde beim virtuellen Test innerhalb des Plant Modells rudimentär durchgeführt und war für die HCU unerheblich. Beim realen Steuergerät benötigte das BMS nun diese Spannungen über CAN aus dem Plant und schickte den SOC wieder zurück. Es musste deshalb eine Erweiterung des Plant Modells durchgeführt werden.

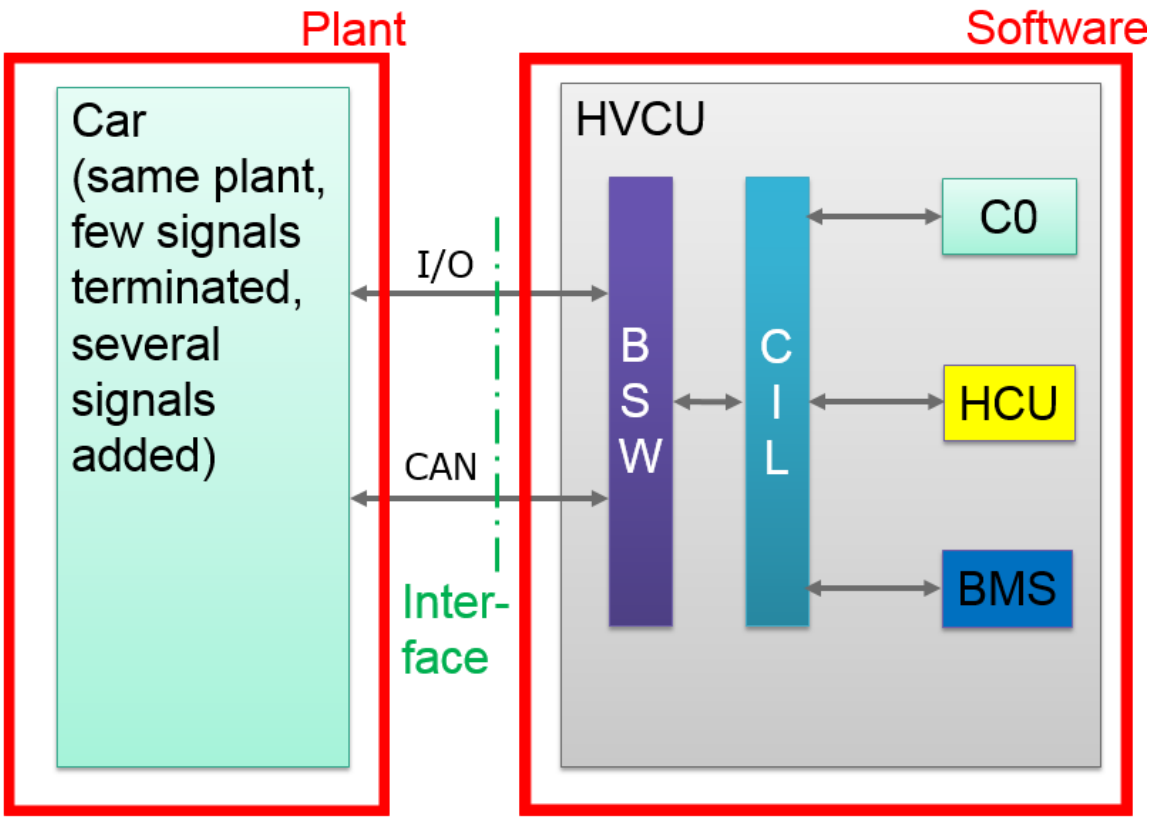

Abbildung 6.1: Testaufbau für den realen Test

Innerhalb der HVCU werden auch Signale zwischen HCU und BMS sowie HCU und C0 ausgetauscht. So ging im vorherigen Testschritt die Drehmomentanforderung der HCU

ans Plant, verblieb nun aber steuergerätintern und wurde von der HCU über CIL an die C0 geschickt. Diese Signale mussten aus dem Plant Modell entfernt werden.

Für eine fehlerfreie Funktion des Steuergeräts mussten alle CAN Signale vorhanden sein. Alle im Plant verfügbaren Signale wurden verwendet und die restlichen gegebenenfalls zusätzlich simuliert, eine sogenannte „Restbussimulation“.

Die HVCU kommunizierte über drei CAN Busse mit anderen Steuergeräten im Fahrzeug, dem Hybrid-CAN, Powertrain-CAN und dem Private-CAN. Zusätzlich kam noch ein LIN Bus zu einem Intelligent-Battery-Sensor (IBS), welcher die Spannung und den Strom der herkömmlichen Fahrzeugbatterie überwachte (Abbildung 6.2). Die beiden ersteren CANs beinhalteten je etwa 90 Signale für die HVCU, der letztere 180.

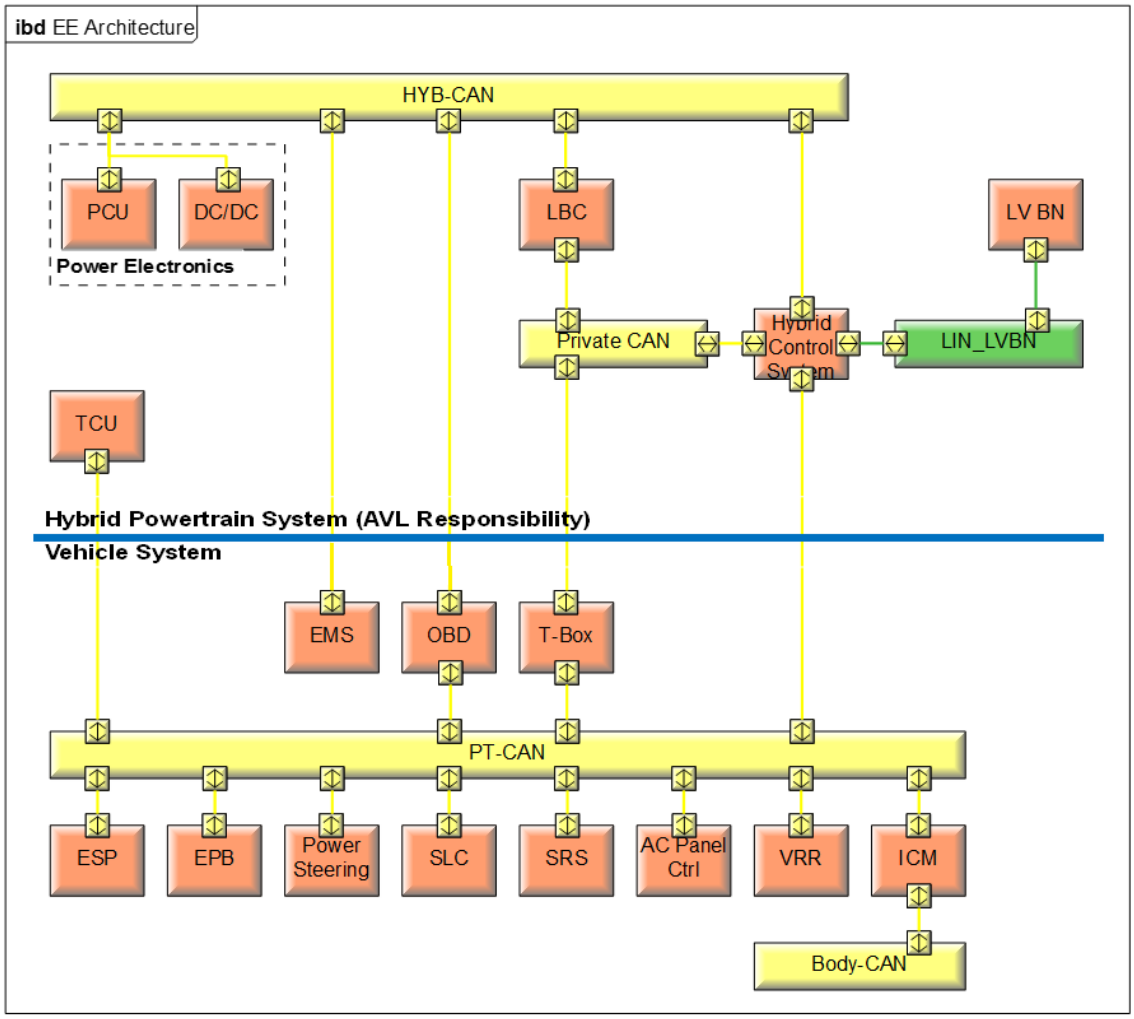

Abbildung 6.2: Konnex der Komponenten und CAN Busse [24]

Das Steuergerät besaß auch Schnittstellen zu Sensoren und Aktoren. Die HCU SW benötigte nur zwei digitale Ausgänge, einen für das Hauptrelais und den zweiten für die

Weitergabe des Signals von Klemme 50 und ein paar Eingänge. Mit Klemme 50 wird die Starterinformation am Starter weitergegeben. Möchte der Fahrer starten, so wird diese Information zuerst direkt an die HVCU übermittelt, welche anschließend entscheidet, ob elektrisch gefahren werden soll oder sie das Signal an das Verbrennungsmotorsteuergerät weitergibt.

Weiters waren einige Komponenten über eine Leitung in einer Signalschleife (HVIL) miteinander verbunden (Abbildung 6.3). Die HVCU war dabei der Ausgangs- und Endpunkt. Sie legte eine Spannung von 5V an, und maß den Spannungspegel des eingehenden Signals. Der gemessene Spannungsabfall durfte nur in einem bestimmten Toleranzband sein, andernfalls schlug die Diagnose an.

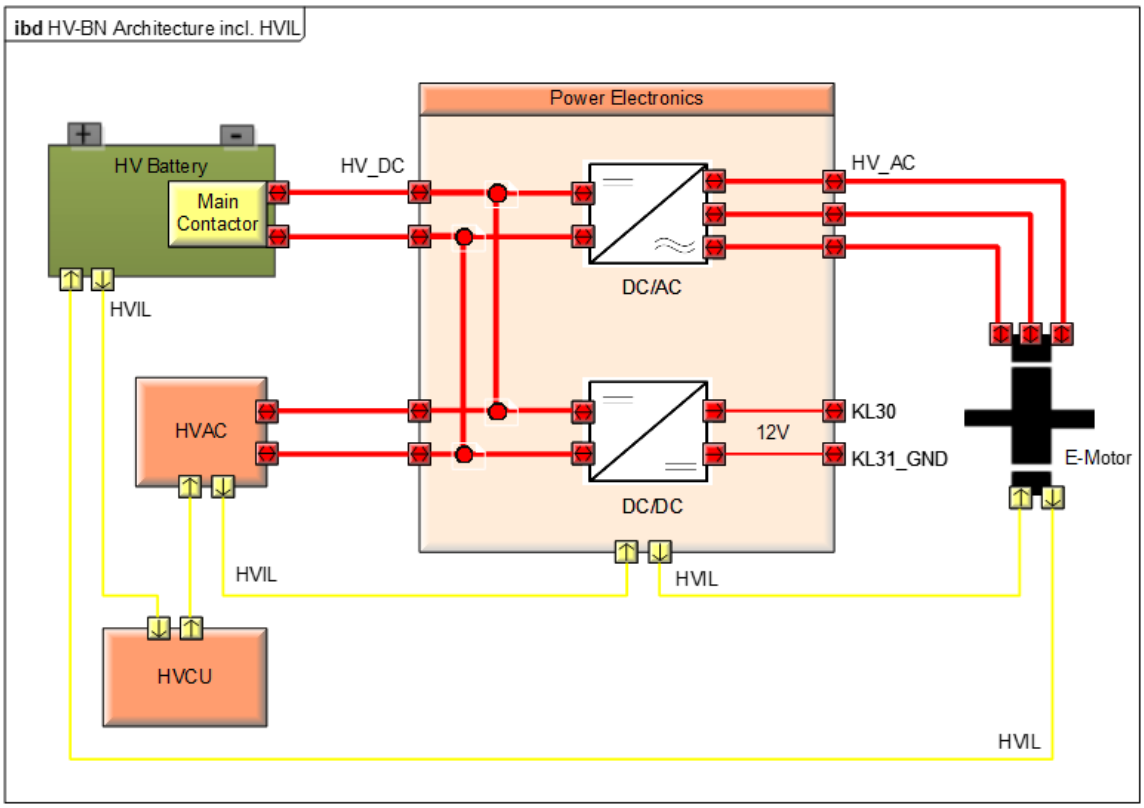

Abbildung 6.3: Komponenten überwacht durch HVIL Signal [24]

Mit diesem Signal wurde rudimentär die Präsenz der Elemente HVAC, der Leistungselektronik, dem Elektromotor und der HV Batterie überprüft.

Die meisten weiteren Pins, etwa 20 an der Zahl, wurden vom BMS oder C0 benötigt und führten zu einer erhöhten Komplexität des Testaufbaus. Es wurden zusätzliche analoge Signale für Temperatur- und Drucksensoren sowie für Schalter mit mehreren

Widerständen in Serie benötigt. Auch mehrere digitale Ein- und Ausgänge für Relais der HV Batterie, Schaltern und Ansteuerungen von Ventilen waren nun notwendig.

6.1 Testanordnung

Das Simulink Modell der HCU wurde in MC durch vier RT Blöcke ersetzt, welche die Kommunikation mit dem Steuergerät übernahmen (Abbildung 6.4).

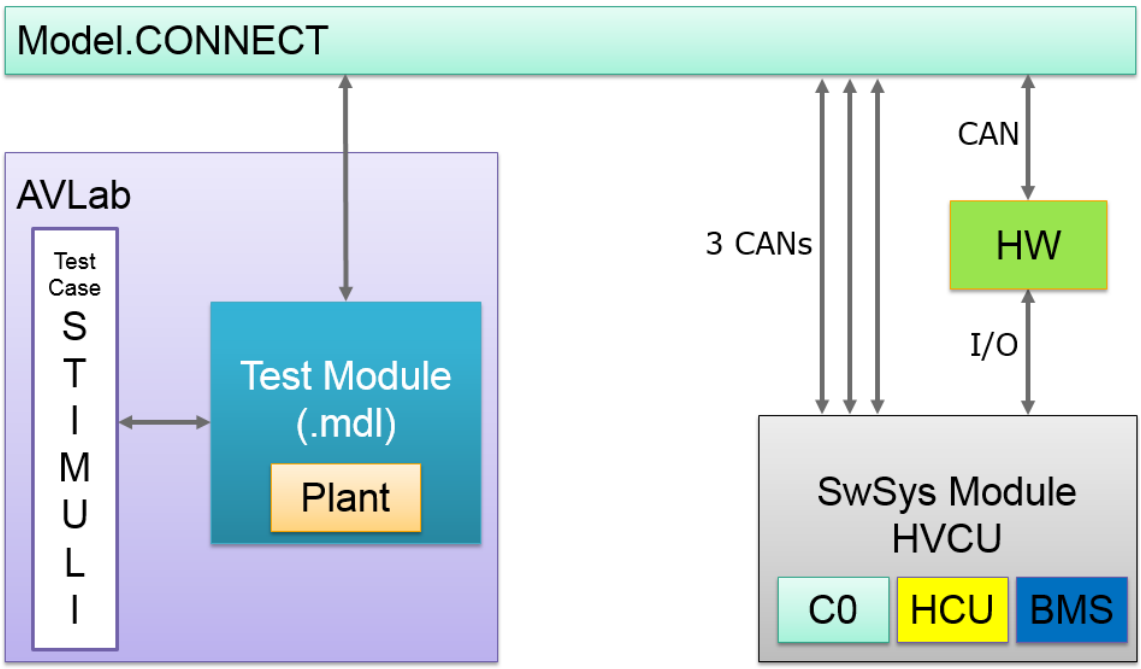

Abbildung 6.4: MC als Schnittstelle zwischen dem Plant Modell und dem Steuergerät

Drei Echtzeitschnittstellen bedienten die CAN Busse der HVCU und eine weitere tauschte Informationen mit den I/O-Simulatoren aus.

6.2 Simulation der Schnittstellen

Um einen Testfall korrekt abzubilden, mussten auch alle Hardwareschnittstellen des Steuergeräts mit den entsprechenden Signalen versorgt werden. Es wäre möglich ein reales Fahrpedal oder Schalter einzubinden, jedoch nicht die exakten Signale für einen Testfall nachzustellen. Somit mussten die Ein- und Ausgänge über Module, die vom Plant Modell gesteuert wurden, versorgt werden.

6.2.1 Analoge Eingänge

Um die Sensoren und Aktuatoren bestmöglich zu simulieren, wurden als erstes alle Parameter der Signale ermittelt. Alle sechs analogen Eingänge der HVCU hatten einen

Spannungspegel zwischen 0 und 5V und wurden mit 12 Bit digitalisiert. Zwei Sensoren, jene für den Tempomatschalter und den NTC Temperatursensor des Kühlkreislaufs, wurden von der HVCU versorgt und ihre Signale über einen externen Spannungsteiler erzeugt (Abbildung 6.5).

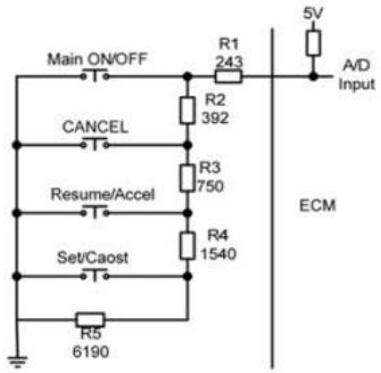

Abbildung 6.5: Widerstandsleiter für die Schalter des Tempomaten [24]

Die Schaltung des Temperatursensors ist ein einfacher Spannungsteiler, der intern einen Pull-Up Widerstand besitzt. Der NTC Thermistor ändert mit der Temperatur seinen Widerstand und somit kann man anhand des Spannungsabfalls auf die aktuelle Temperatur schließen. An diese beiden Eingänge konnte man nicht direkt die gewünschte Signalspannung anlegen, sondern musste einen Impedanzwandler dazwischen schalten. Bei den anderen vier Signalen konnte man direkt eine analoge Spannung anlegen.

Hier benötigte man sechs analoge Ausgänge für einen Spannungsbereich von 0 bis 5V mit einer Auflösung von mindestens 12 Bit und zusätzlich zwei Impedanzwandler. Weitere zwei analoge Eingänge (HVIL) werden im Kapitel 6.4.3 abgehandelt. Für den korrekten Spannungsabfall sorgte da ein einfacher ohmscher Widerstand.

6.2.2 Digitale Eingänge

Es gab fünf digitale Eingänge, von denen vier gegen die Versorgungsspannung geschalten wurden. Sie erfassten die Betätigung des Bremspedals und den Status der Klemmen 15 und 50. Ein Eingang musste für die Aktivierung gegen Masse geschalten werden. Er war mit der EV-Taste gekoppelt, mit welcher der Fahrer den Wunsch nach rein elektrischem Antrieb äußern kann. Für diese Eingänge benötigte man vier *Highside*-Ausgänge und einen *Lowside*.

6.2.3 Analoge Ausgänge / PWM Ausgänge

Die Bezeichnungen für die drei *Lowside*-Schalter der Schütze wurden fälschlicherweise als pulsweitenmodulierte Signale ausgewiesen, sind jedoch gewöhnliche, nach Masse schaltende Digitalausgänge.

Für die Ansteuerung der beiden Magnetventile im Hydraulikmodul der C0 Kupplung wurde ein pulsweitenmoduliertes Stromsignal ausgegeben. Dieses schaltete mit einer Frequenz von bis zu 3000Hz einen Strom bis 2A, und blieb dabei an der Versorgungsspannung mit einer Toleranz von etwa 20%. Da die direkte Messung solch eines Stroms nur sehr kostenintensiv zu realisieren war, wurde eine indirekte Messung über einen Messwiderstand vorgenommen. Dazu gehörte auch eine Verstärkung des Signals, welche über einen nichtinvertierenden Verstärker vorgenommen wurde.

Es waren zwei analoge Eingänge erforderlich, welche eine hohe Auflösung hatten. Der Eingangsspannungsbereich war abhängig von der Verstärkung des Signals.

6.2.4 Digitale Ausgänge

Letztendlich hatte das Steuergerät noch sechs digitale Ausgänge, von denen die Hälfte gegen Masse und die andere gegen die Versorgungsspannung schalteten.

Es waren jeweils drei Digitaleingänge mit Pull-up und Pull-down Beschaltung notwendig.

6.3 PEAK Module (MicroMods)

Da diese I/O-Signale auch über MC ausgegeben und eingelesen wurden, lag es nahe Module zu verwenden, mit welchen man über einen CAN Bus kommunizieren kann.

Sogenannte CAN Knoten werden von zahlreichen Herstellern angeboten, beispielsweise von National Instruments. Es gibt hochpreisige Module von NI in zwei grundsätzlichen Varianten, bei denen man über Einschubkarten beliebige Funktionen hinzufügen kann, wie eine CAN Schnittstelle oder analoge Ausgänge mit einer breiten Auswahl an Ausgangsspannungsbereichen und Auflösungen. Bei der ersten ist im Gerät ein Hochleistungsprozessor verbaut auf dem ein Echtzeit-Linux Betriebssystem ausgeführt wird und auf dem etwa ein Plant Modell simuliert werden kann. Die Datenübertragung zum Computer stellt dann nur eine Möglichkeit zur Auswertung und Visualisierung der Daten dar. Bei der zweiten Spielart dient das NI Modul nur als I/O Schnittstelle und besitzt keinerlei Intelligenz. Beide Systemvarianten wurden aufgrund preislicher Überlegungen verworfen.

Als Alternative wurden Produkte von MRS Electronic genannt, welche bereits bei AVL List GmbH erfolgreich verwendet werden und speziell für den Einsatz in und um das KFZ angedacht sind. Aufgrund fehlender Kompatibilität mit unseren Anforderungen, wie unzureichende Auflösung oder mangelhafte Beschaltungsmöglichkeit der Digitalausgänge wurde auch dieses Angebot ausgeschlagen.

Erfreulicherweise stellte die Firma PEAK Systems, deren USB CAN Adapter von MC unterstützt wurden, auch I/O-Module her. Durch Kombination von drei Modulen, einem *PCAN-MicroMod Mix 3* für die digitalen I/Os und zwei *PCAN-MicroMod Analog 2* für die analogen, konnte die gesamte Beschaltung der HVCU bedient werden, somit wurden diese drei Module angeschafft. Im aktuellen Projekt wurden nicht alle Schnittstellen verwendet und boten Platz für Erweiterungen hinsichtlich der Funktionen des Steuergeräts, sowie eine ausreichende Reserve für den Einsatz in Nachfolgeprojekten.

Man konnte alle Module über einem CAN Bus verbinden und aus MC ansprechen. Dafür wurde eine eigene CAN Initialisierungsdatei geschrieben, um die Signale aus Simulink einer CAN Nachricht und einem CAN Signal zuzuordnen. In den Modulen musste anschließend jedes Signal einem Ausgang oder Eingang zugewiesen werden, die Konfiguration erfolgte dabei nach dem Handbuch.

6.4 Schaltung zwischen MicroMods und HVCU

Es war wichtig, eine genaue Zuordnung zwischen den Schnittstellen des Steuergeräts und den Modulen herzustellen. Die drei *MicroMods* wurden teilweise mit einer Zusatzbeschaltung mit dem Steuergerät verbunden.

6.4.1 MicroMod Mix 3 #0

Mit diesem Modul wurden die digitalen Ein- und Ausgänge des Steuergeräts bedient. Da das Modul mit 12V versorgt wurde, und die Eingänge der HVCU auch diesen Spannungslevel als *High* erkannten, konnte eine direkte Verbindung hergestellt werden. Der Schaltplan des Aufbaus ist in Abbildung 6.6 gezeigt.

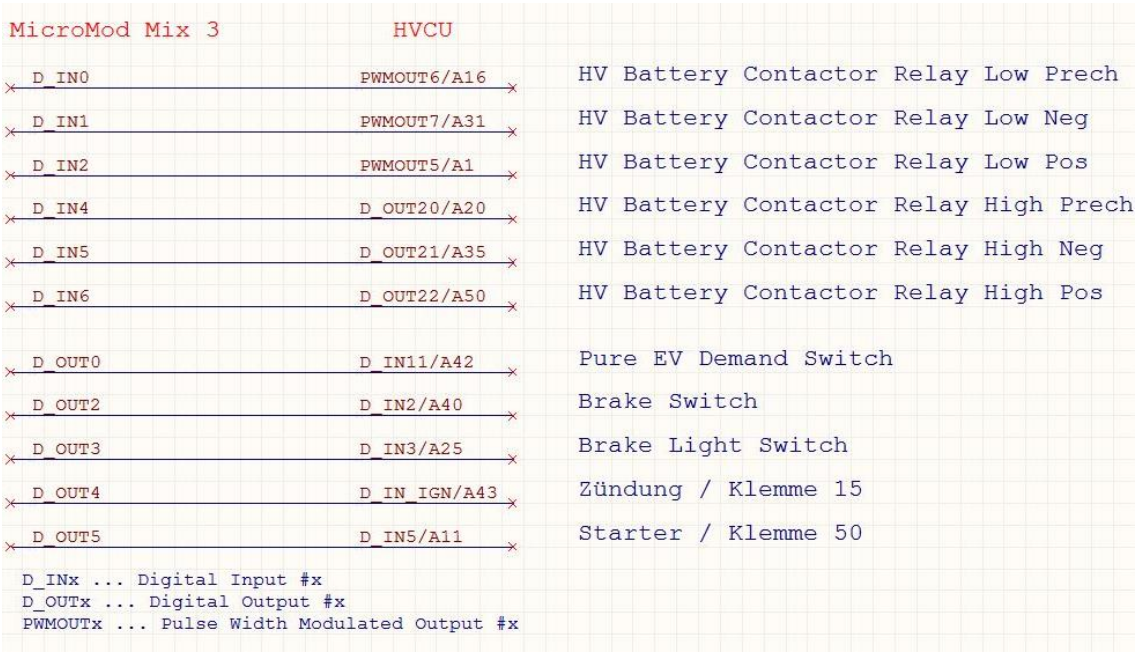

Abbildung 6.6: Anschluss des MicroMod Mix 3 mit der HVCU

Auf der linken Seite sind die I/O-Bezeichnungen des *MicroMods* zu sehen, auf der rechten die zugehörigen Klemmennummern und die Namen der Signale in der Software. Auch die Funktion im Fahrzeug ist beschrieben. Es musste keine Änderungen an den Signalen durchgeführt werden und somit konnten alle Klemmen direkt verbunden werden.

6.4.2 MicroMod AN #1

Für die analogen Eingänge und die PWM Ausgänge war eine zusätzliche Beschaltung erforderlich. Wie in Kapitel 6.2.1 beschrieben, wurden je ein Impedanzwandler und zusätzlich ein Längswiderstand vor zwei Eingängen geschalten. Zusätzlich erhielten alle analogen Eingänge der HVCU, welche nur einen zulässigen Eingangsspannungsbereich von 0 bis 5V hatten, eine Schutzbeschaltung gegen Überspannung in Form einer Zenerdiode, da es den Analogmodulen möglich war, bis zu 10V auszugeben.

Für die beiden Stromsignale der Ventile für die Kupplung wurden nichtinvertierende Verstärker [25] realisiert, der Schaltplan ist in Abbildung 6.7 gezeigt.

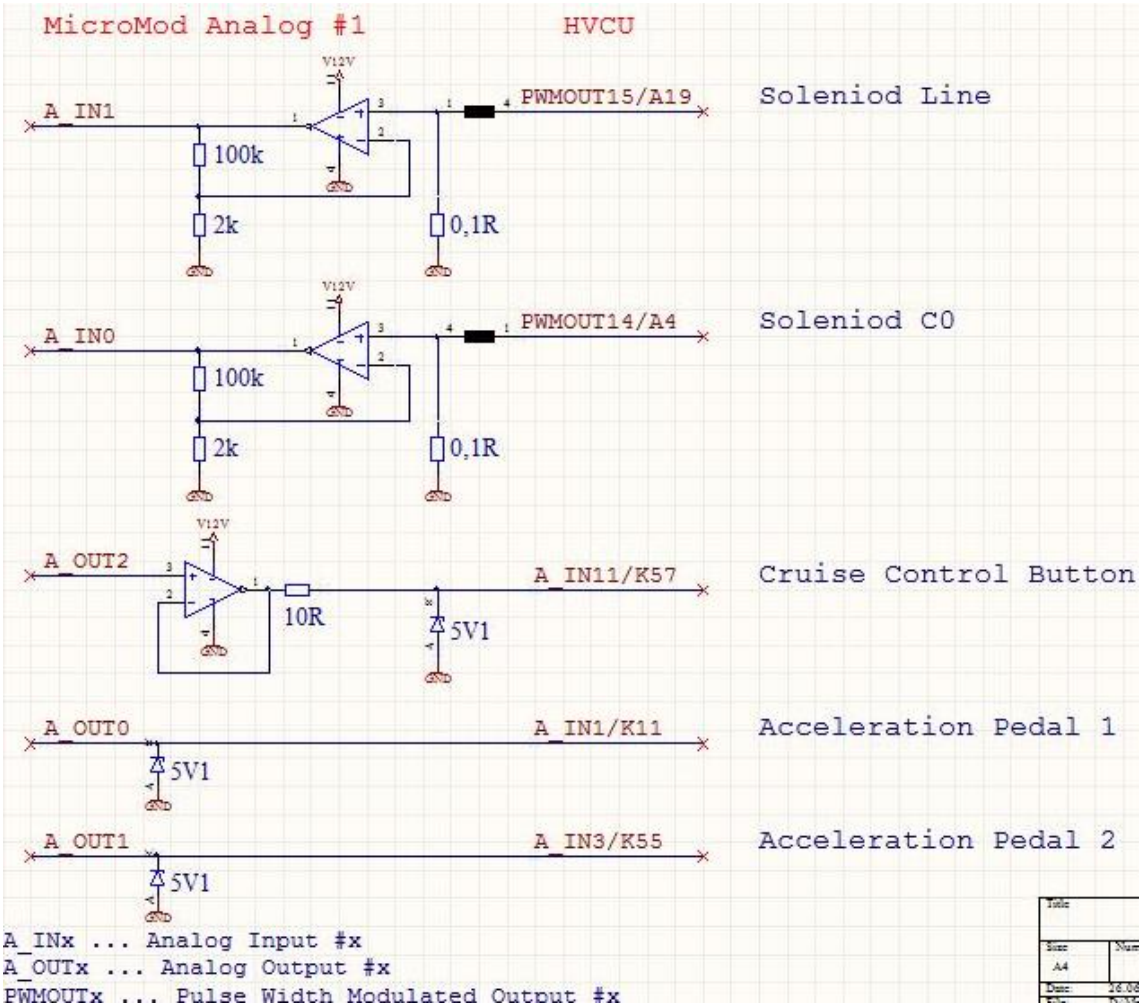

Abbildung 6.7: Schaltplan der Zusatzplatine

6.4.3 MicroMod AN #2

Der zweite *MicroMod* für die analoge I/O verwendete nur drei analoge Ausgänge, die wieder eine Schutzbeschaltung in Form einer Zenerdiode aufwiesen (Abbildung 6.8).

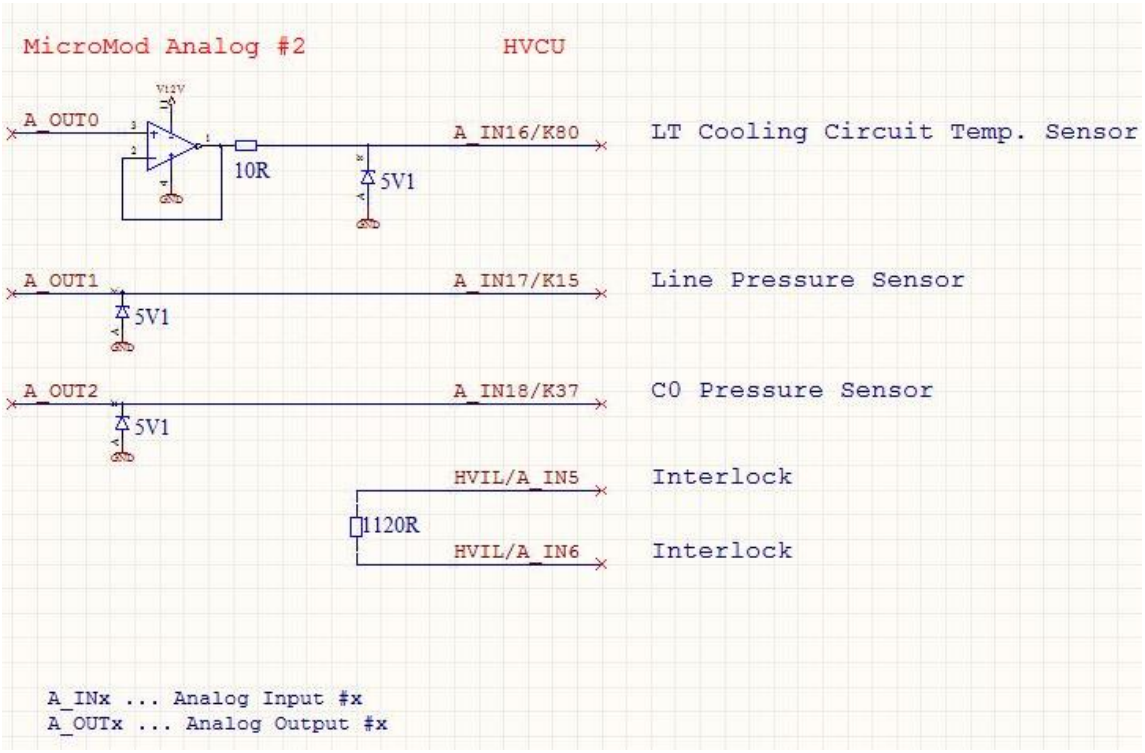

Abbildung 6.8: Beschaltung für den MicroMod Analog #2

Zusätzlich ist in dem Schaltplan die Beschaltung der Interlock Signale (HVIL) ersichtlich. An einem Port wurde eine Spannung von 5V ausgegeben, und am anderen musste für ein gültiges Signal eine Eingangsspannung zwischen 1,75V und 1,4V anliegen. Mit dieser Anforderung und einem kleinen Strom wurde ein Widerstand nach (1) berechnet:

$$R_{interlock} = \frac{U_{supply} - U_{sig}}{I_{interlock}} = \frac{5 - 1{,}6}{0{,}003} \approx 1120\,\Omega \quad \textbf{(1)}$$

$R_{interlock}$ ... Widerstand zwischen den HVIL Kontakten

$I_{interlock}$ ... Strom zwischen den HVIL Kontakten

U_{supply} ... Interne Versorgungsspannung des Steuergeräts

U_{sig} ... Spannung am HVIL Eingang für eine fehlerfreie Funktion

Es wurde auf $1120\,\Omega$ gerundet, da es die Kombination aus zwei verfügbaren Widerstandswerten war, und diese in den gängigen Widerstandsreihen vorkommen.

6.4.4 Platine

Die Zusatzbeschaltung zwischen den Modulen und der HVCU wurde auf einer Lochrasterplatine aufgebaut und ist in Abbildung 6.9 zu sehen. Auf der linken Seite

befinden sich die beiden nichtinvertierenden Verstärker und die Messwiderstände, auf der rechten sind die beiden Spannungsfolger und die Zenerdioden der analogen Ausgänge zu sehen.

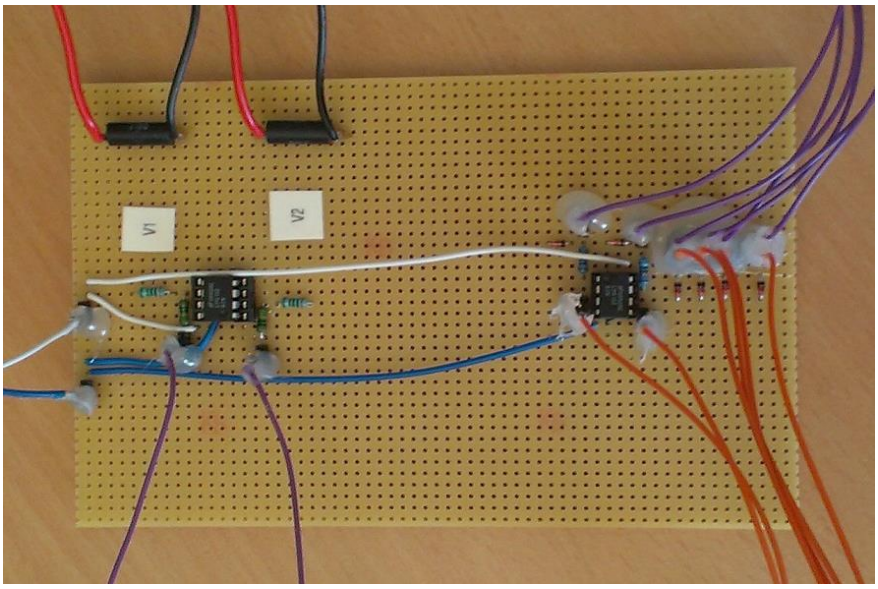

Abbildung 6.9: Zwischenplatine

Die orangen Drähte sind die Eingänge der Platine und kommen von den *MicroMods*, die violetten sind Ausgänge. Auf der rechten Seite werden die Ausgangsdrähte ans Steuergerät angeschlossen, auf der linken mit den analogen Eingängen der *MicroMods*. Bei den Messwiderständen markieren die schwarzen Drähte das Bezugspotential und die roten werden an den negativen Kontakt der Ventile angeschlossen. Blaue (Masse) und weiße (12V) Drähte markieren die Anschlüsse für die Versorgung der Mikrochips. Das Bezugspotential der Elektronik und die der Messwiderstände wurde aus EMV Gründen nicht zusammengelegt.

In dem Mikrochip LM6132 von Texas Instruments befinden sich zwei Operationsverstärker (Abbildung 6.10). Er erfüllte die Forderungen nach einer Versorgungsspannung von 12V und möglichen Ausgangsspannungen bis 10V.

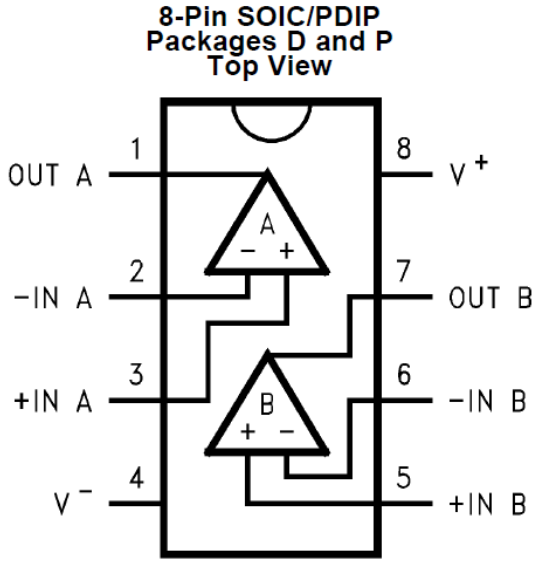

Abbildung 6.10: Interner Aufbau und Pinbelegung des LM6132 [26]

Es wurde ein 8-Pin PDIP Gehäuse mit Sockel verwendet, um eine einfache Austauschbarkeit des Chips zu gewährleisten.

6.5 Strecke des Druckregelkreises außerhalb des Plant Modells

Betrachtet man die HVCU und das Plant als Regelkreis, also dargestellt als Regler und Strecke, so ist die Strecke für alle Signale aus der HCU vollständig, nicht jedoch für die C0 aufgrund des schon vorhin beschriebenen Umstands der Schnittstellenverschiebung.

In Abbildung 6.11 sind auf der rechten Seite die Signale zwischen der HVCU (Hybrid Control System) und dem Kupplungsmodul C0 (P2 Hybrid Module) zu sehen. Auf der linken Seite befinden sich oben die Signale für die Schütze der HV Batterie, welche im nächsten Kapitel eingehender Betrachtung bedarf. Das Signal *HVBatCtctr_Ctrl* steht dabei für alle drei elektrisch betätigten Schalter.

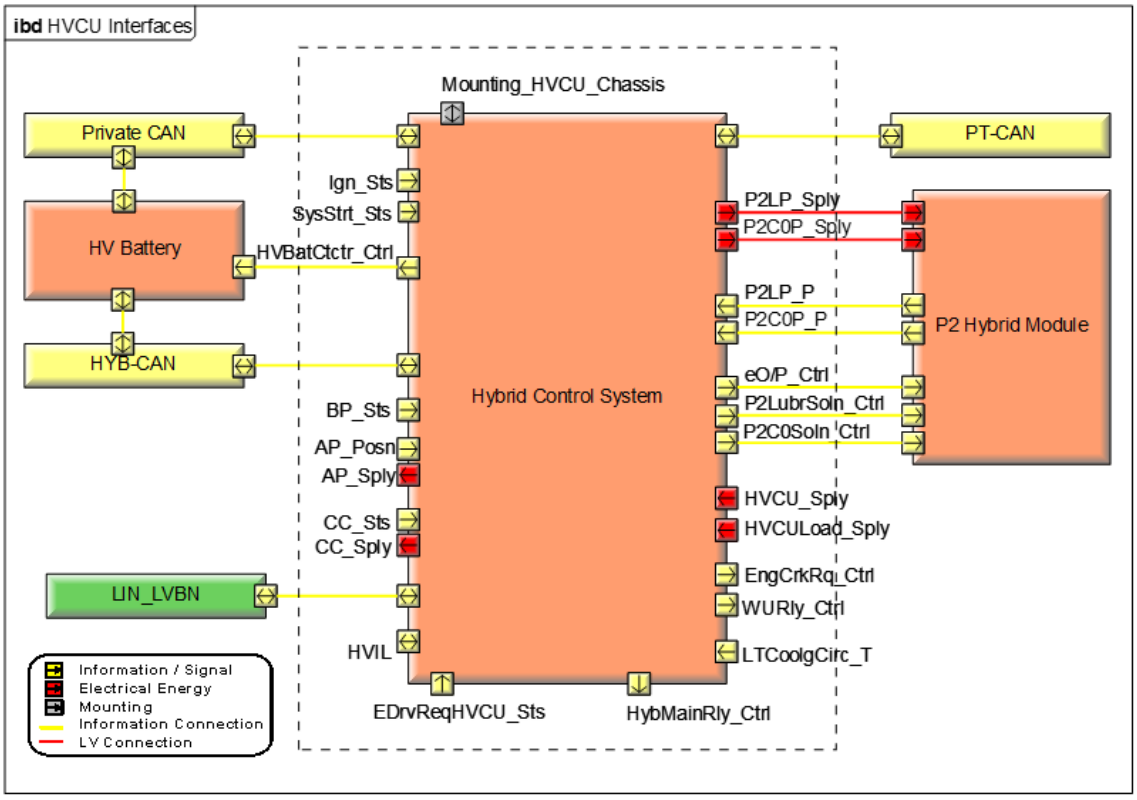

Abbildung 6.11: HVCU und verbundene Komponenten [24]

Das Funktionsprinzip für die hydraulische Kupplungsansteuerung ist in Abbildung 6.12 dargestellt. Eine elektrische Ölpumpe baut Line-Druck und Lube-Druck auf. Der Strom durch den C0 Solenoid im 3/2-Wegeventil *CLUTCH VFS* (rechts im Bild) gibt den Bedarf an Druck für die Kupplung an, der übrige Druck ist für Lube (Schmierung von C0). Die Höhe des Schmierungsdrucks ergibt sich über das elektrisch gesteuerte 3/2-Wegeventil *LUBE MDA NH* (mittig im Bild), welches abhängig vom Strom und dem Line-Druck den Steuerdruck für das 2/2-Wegeventil *LUBE SPOOL VALVE* (links im Bild) vorgibt. welches den Durchfluss für die Schmierung freigibt.

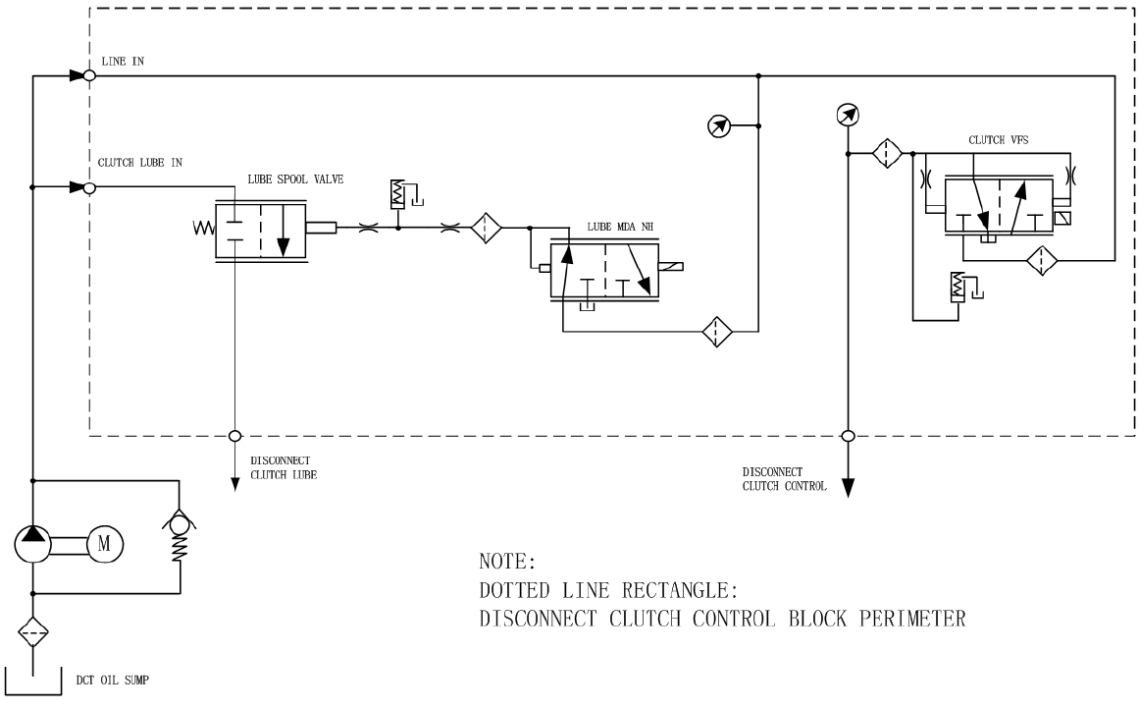

Abbildung 6.12: Hydraulischer Schaltplan der Kupplungsansteuerung [24]

Das Prinzip ist gleich einem Transistor. Der Druck vom mittleren zum linken Ventil ähnelt dem Steuerstrom, welche dann abhängig von seiner Größe den Strom (Druck) durch den Hauptpfad steuert.

Die anderen Kupplungen, welche sich im Doppelkupplungsgetriebe befinden, werden mechanisch angetrieben und nicht von der HVCU geregelt.

Die beiden Ventile wurden mit einem pulsweitenmodulierten Stromsignal angesteuert und als reale Aktoren mit dem Steuergerät verbunden. Für die Simulation des aufgebauten Drucks musste nun dieser Strom gemessen, und anhand einer Kennlinie in einen entsprechenden Druck umgewandelt werden. Dieser wurde im Fahrzeug gemessen und als analoges Spannungssignal an einen HVCU-Eingang gelegt.

Um den Strom durch die Ventile zu messen, wurde ein Messwiderstand in Serie in den Strompfad geschaltet und der Spannungsabfall gemessen. Die Ventile wurden mit einem Strom bis zu 2A angesteuert, wobei die Spannung zwischen 9,8V und 15V schwanken durfte.

Der Analogeingang eines *MicroMod Analog 2* hatte einen Eingangsspannungsbereich von 0 bis 10V. Zuerst wurden Überlegungen zum maximalen Spannungsabfall am Shunt bezüglich Beeinflussung des Messobjekts angestellt, und anschließend die Verstärkung

berechnet. Bei dem Maximalstrom und einem Standardwiderstandswert wurde nach (2) ein vernachlässigbar kleiner Spannungsabfall berechnet.

$$U_{shunt,max} = R_{shunt} * I_{max} = 0{,}1\Omega * 2\text{A} = 0{,}2\text{V} \quad (2)$$

$U_{shunt,max}$ … Maximalspannung am Messwiderstand

R_{shunt} … Größe des Messwiderstands

I_{max} … Maximaler Strom durch Messwiderstand

Auch die maximale Verlustleistung $P_{v,max}$ hielt sich bei diesem Wert in Grenzen (3), da $0{,}4W$ für einen großen bedrahteten Widerstand annehmbar sind.

$$P_{v,max} = R_{shunt} * {I_{max}}^2 = 0{,}1\Omega * (2\text{A})^2 = 0{,}4\text{W} \quad (3)$$

$P_{v,max}$ … Maximale Verlustleistung am Messwiderstand

Die Spannung wurde mit einem nichtinvertierenden Verstärker auf den Eingangsspannungsbereich des *MicroMod Analog 2* angepasst (4).

$$U_{mess,max} = V * U_{shunt,max} = \left(1 + \frac{R_2}{R_1}\right) * U_{shunt,max} = \left(1 + \frac{100k\Omega}{2{,}2k\Omega}\right) * 0{,}2V \approx 9{,}3V \quad (4)$$

$U_{mess,max}$ … Maximale Spannung am Messwiderstand

V … Verstärkungsfaktor

R_1 … Größe des Widerstands zwischen Bezugspotential und negativem Eingang

R_2 … Größe des Widerstands zwischen Ausgang und negativem Eingang

Dieser schickte anschließend über CAN das mit 16 Bit digitalisierte Signal an MC.

In Abbildung 6.13 wird die Strecke für den Kupplungsdruck gezeigt. Über die MC Schnittstelle wurde die am Shunt gemessene Spannung ins Simulink Modell als 16 Bit-Wert gelesen und in der nachfolgenden Lookup-Table daraus der entsprechende Spannungswert ermittelt, mit welchem anschließend über die Verstärkung und der Größe des Shunts auf den Strom nach Formel (5) geschlossen werden konnte.

$$I_{solenoid} = \frac{U_{shunt}}{R_{shunt}} = \frac{U_{mess}}{V} * \frac{1}{R_{shunt}} \quad (5)$$

$I_{solenoid}$ ... Strom durch das Ventil

U_{shunt} ... Spannung am Messwiderstand

U_{mess} ... mit dem MicroMod gemessene, verstärkte Spannung am Shunt

Dieser Strom entsprach einem Hydraulikdruck in der Kupplung, welcher durch Messungen am Fahrzeug ermittelt und in eine weitere Lookup-Table verpackt wurde.

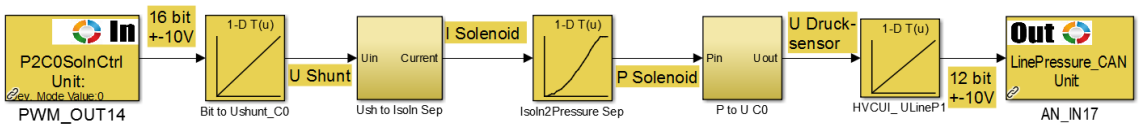

Abbildung 6.13: Strecke der Abhängigkeit zwischen Strom und Aktivierungsdruck in der Kupplung

Die Ausgangsspannung des Drucksensors war nicht nur abhängig vom Druck, sondern auch von der Versorgungsspannung und wurde nach (6) berechnet. Diese Formel konnte im Datenblatt nachgeschlagen werden.

$$U_{pressuresensor} = (0{,}04379 * p - 0{,}005528) * U_{supply} \quad (6)$$

$U_{pressuresensor}$ ... Ausgangsspannung des Drucksensors

p ... Hydraulikdruck in der Kupplung

U_{supply} ... Versorgungsspannung des Sensors

Als letzter Schritt musste nun nur mehr die Ausgangsspannung des Drucksensors in einen 12 Bit Wert umgerechnet werden, um ihn anschließend über MC an den *MicroMod Analog 2* zu schicken und als Spannungswert ausgeben zu lassen.

Die Rückkopplung des Schmierungsdrucks erfolgte etwas anders. Da diese Masterarbeit Teil eines laufenden Kundenprojekt war, gab es immer wieder Änderungen am System und dessen Umsetzungen. So funktionierte der erste Entwurf der Schmierung aufgrund von einer großen Hysterese in der Strom-Druck Beziehung nicht richtig und wurde später anders umgesetzt. Hier wurde, weil es mit der verwendeten Steuergeräte-Software

zusammenpasste, das ursprüngliche Konzept umgesetzt, welches nur eine provisorische Lösung darstellte und in Abbildung 6.14 zu sehen ist.

Der gemessene Spannungsabfall am Shunt im Schmierungsdruck-Strompfad wurde wieder als 16 Bit Wert eingelesen und in den Steuerstrom umgewandelt. Aufgrund des schlechten Verhaltens wurde aber immer 0,74A ausgegeben, und der Druck tatsächlich nur über die Ölpumpendrehzahl geregelt.

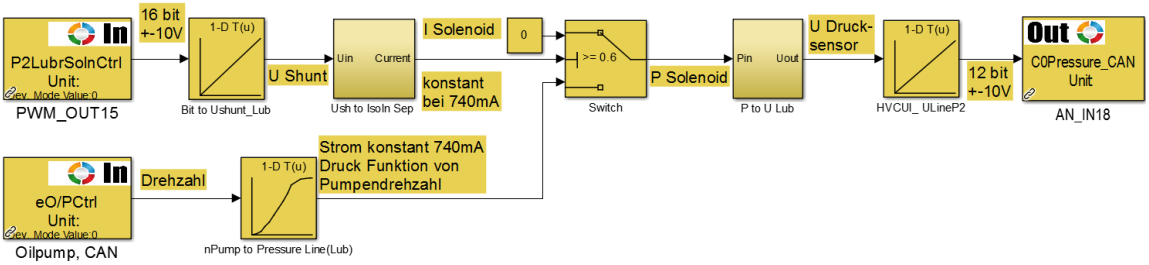

Abbildung 6.14: Strecke der Abhängigkeit zwischen Strom und Schmierungsdruck in der Kupplung

Mit der unteren *ICOS* Schnittstelle wurde die Pumpendrehzahl eingelesen und anschließend in einen Druck übergeführt. Der nachfolgende Schalter setzte nur bei aktivem Stromsignal diesen Druck gültig, andernfalls war er null. Anschließend folgte, gleich wie vorhin, die Umwandlung des Drucks mit dem simulierten Drucksensor in ein Spannungssignal.

6.6 Konvertierung analoger Signale

Die Signale aus dem Plant, welche beim Steuergerät auf analoge Eingänge gelegt wurden, mussten in Simulink in ein 12 Bit Signal umgewandelt werden. Dies betraf etwa die beiden Spannungen für das Fahrpedal (Abbildung 6.15).

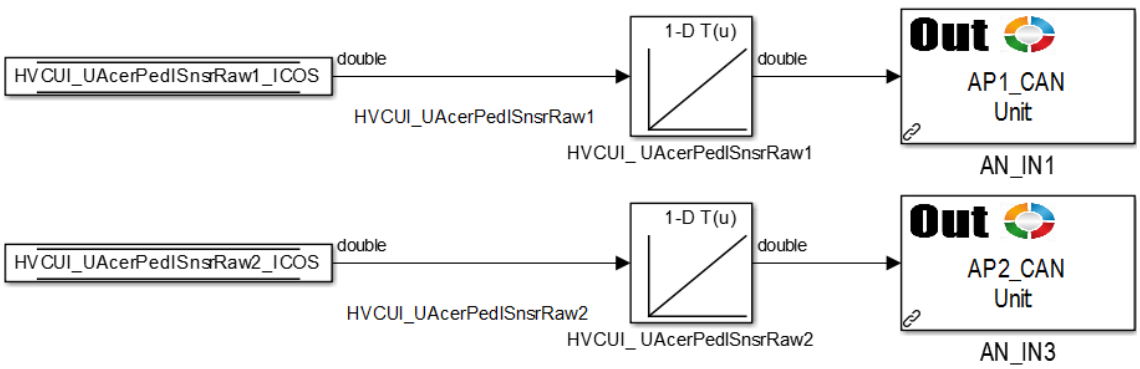

Abbildung 6.15: Konvertierung von analogen Signalen in digitale Signale

Auch für den Temperatursensor der Kühlflüssigkeit (*Coolant Temperature*) sowie die Schalter für den Tempomaten (*Cruise Control*) gab es diese Umwandlung.

6.7 Schützkontakte der Hochvoltbatterie

Es gibt drei Schützkontakte, um die HV Batterie mit dem restlichen HV System des Fahrzeugs zu verbinden. Diese werden in der HVCU jeweils gegen Versorgung und Masse geschalten, wie eine Halbbrücke. Das heißt, es sind im Steuergerät sechs Schalter verbaut, drei *Highside* (HS) und drei *Lowside* (LS). Damit nun ein Schütz schließen kann, mussten jeweils ein LS und der dazugehörige HS geschalten werden (Abbildung 6.16).

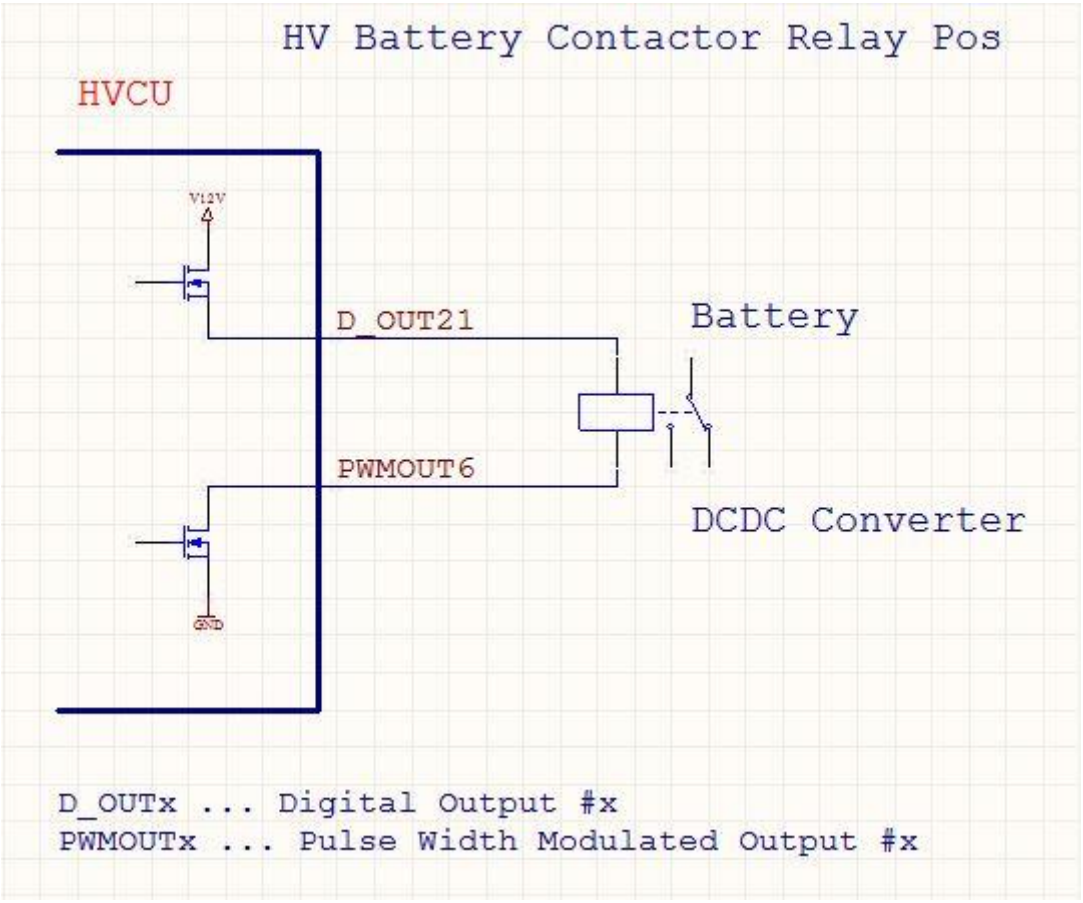

Abbildung 6.16: Beschaltung der Schütze

Durch diesen Aufbau kann die Sicherheitsstufe (ASIL) des HV Systems erhöht werden, es gibt zwei Schalter die den Hochvoltkreis unterbrechen können.

Die Simulation dieser Schütze ist in Abbildung 6.17 dargestellt. Wenn die HVCU nun beispielsweise für den positiven Schütz die HS und LS Schalter betätigt, wird der Status des nun geschlossenen Stromkreises wieder zurück an die HVCU geschrieben.

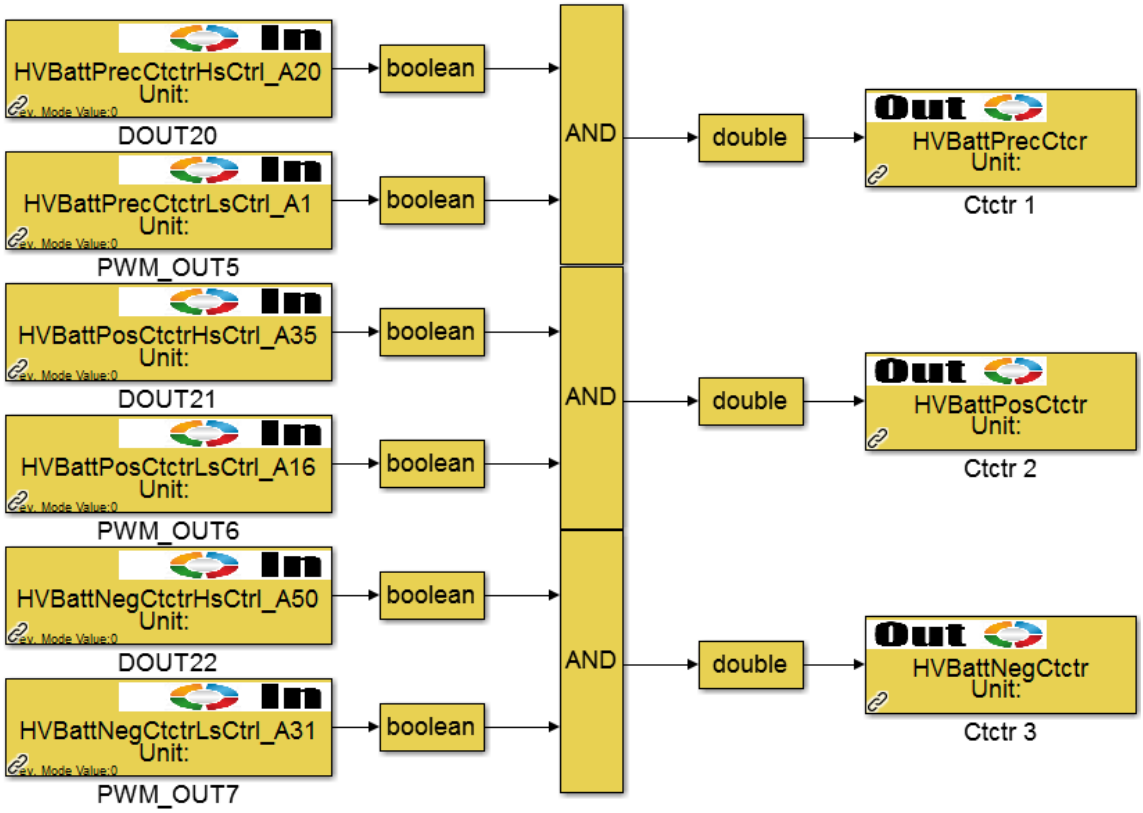

Abbildung 6.17: Simulation der Schützkontakte in Simulink

6.8 Aufbau in MC

Es gab mehrere Ansätze, wie man den Simulationsaufbau umsetzen könnte. In MC benötigte man zumindest vier RT-Blöcke, einen für jeden CAN Bus, und eine MATLAB Schnittstelle zum Plant Modell.

6.8.1 Erster Aufbau mit mehreren Modellen

Die Signale im *SwSys_Test*, und somit im Plant stimmten mit jenen der HCU SW überein. Da nun aber die Kommunikation nicht mehr innerhalb eines MATLAB Workspaces stattfand sondern über CAN, mussten die Signalnamen angepasst werden.

Es wurden Simulink Modelle erstellt, eines für jeden CAN, in denen einfach nur die Simulink-Signalnamen in CAN Signalnamen umgewandelt wurden. Es entstanden vier

zusätzliche Modelle und ein übersichtlicher Aufbau in MC. Die automatische Erstellung dieser Simulink Erweiterung für den Co-Simulationsaufbau wurde durch die vielen einzelnen Modelle erschwert und somit wurde diese Idee wieder verworfen. Es ergab sich auch ein erhöhter Bedarf an Arbeitsspeicher, da MC für jedes eingebundene Simulink Modell eine eigene MATLAB Instanz öffnete. Eine Verlangsamung der Berechnung war bei Simulation auf einem Computer mit Mehrkernprozessor nicht zu erwarten, da MC die Rechenleistung für jedes Modell auf einen eigenen Kern verteilte, natürlich abhängig von der Berechnungsreihenfolge.

6.8.2 Aufbau mit einem Modell

Somit blieb nur die Option, die Signalnamen gleich im Testmodell zu konvertieren, was sich als die beste Lösung herausstellte und auch im finalen Systemaufbau so umgesetzt wurde (Abbildung 6.18).

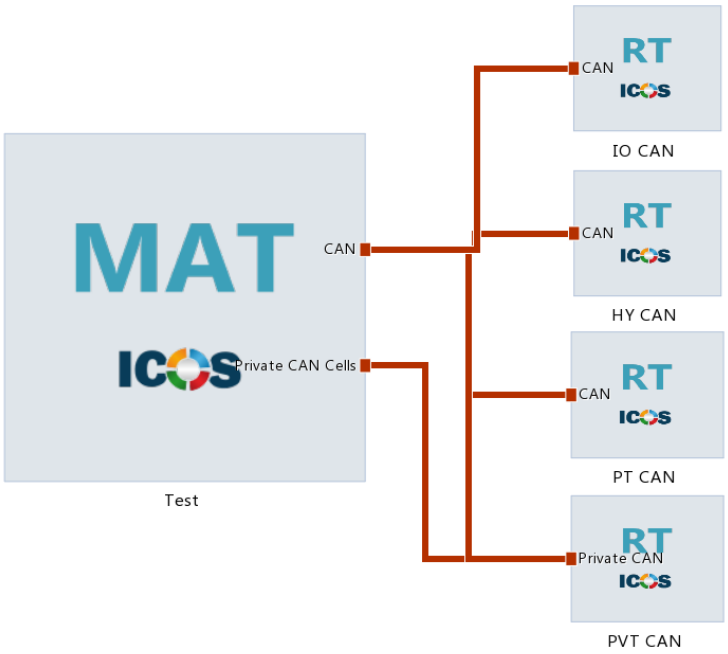

Abbildung 6.18: Aufbau des realen Tests in MC

6.8.3 Laufzeit

Die Simulation musste in Echtzeit ablaufen, da ein Teil auf dem realen Steuergerät gerechnet wurde. Eine Bedingung dafür war, dass das Plant Modell deutlich schneller ausgeführt wurde, um Zeitreserven für die Übertragung der Daten von Simulink nach MC und von MC über CAN an die HVCU zu haben.

Diese Forderung war nicht ganz einfach umzusetzen, da das Fahrzeugmodell mit 1ms gerechnet wurde, um Stick-Slip Effekte der Kupplung so gut als möglich nachzustellen. Mit Abstrichen in der Genauigkeit und vermehrter Schwingungsneigung wurde schließlich das Plant Modell soweit geändert, dass es für sich alleine einen Echtzeitfaktor von 0,28 erreichte. Das heißt, für 10s Simulationszeit dieses Modells wurden 2,8s Rechenzeit benötigt.

Eine weitere Rechenzeiteinsparung erhielt man durch Änderung der Kompilereinstellungen von *Normal* auf *Accelerator*. Diese Einstellung findet man im Simulink Modell rechts neben der Simulationszeit (Abbildung 6.19). Die Auswahl *Rapid Accelerator* brachte keinerlei Verbesserungen, da das Plant als Kernstück ein AVL CRUISE Modell hatte, welches bereits als Library gepackt war. Der Nachteil war allerdings, dass das Modell länger zum Kompilieren benötigte.

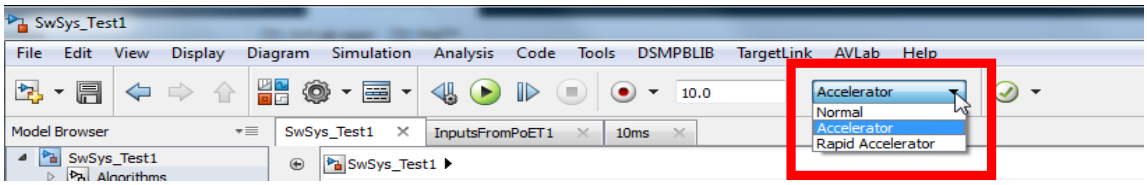

Abbildung 6.19: Beschleunigung der Simulation in Simulink

Jedes Signal in Simulink wurde über einen eigenen *ICOS* Block mit MC verbunden, daraus ergaben sich um die 380 Schnittstellen. Jeder Block führt für die Datenübertragung eine eigene S-Funktion aus und überträgt die Signaldaten. Zur Auslagerung dieser rechenintensiven Signalsynchronisierung wurden alle Ausgangssignale sowie Eingangssignale zu einem Vektor zusammengefasst, um pro Makroschritt nur einmal diese Funktion öffnen zu müssen und alle Signaldaten auf einmal zu synchronisieren.

In einem Testprojekt wurden insgesamt 600 Signale zwischen zwei Simulink Modellen über MC ausgetauscht. Bei der Übertragung der Einzelsignale mit eigener *ICOS* Schnittstelle betrug die Rechenzeit 33s für 10s Simulationszeit. Packte man je 300 Signale auf einen Vektor, benötigte man nur zwei *ICOS Ports*, einen *In* und einen *Out*, und die Rechenzeit verkürzte sich auf 3s.

Das Problem dabei ist, dass die RT-Blöcke in MC nicht mit Vektoren umgehen können, da die Signalnamen beim Vektor verloren gehen. Eine Möglichkeit diese Schwierigkeit zu umgehen wäre, dass man ein weiteres Simulink Modell erstellt, welches einen Vektor wieder in die einzelnen Signale aufspaltet. Die direkte Verwendung von Vektoren in RT-

Blöcken wurde bei Erstellung dieser Arbeit nur für die UDP Schnittstelle unterstützt, nicht jedoch für CAN.

In MC gab es ebenfalls ein paar Möglichkeiten die Rechenzeit zu verkürzen. Das Simulink Modell rechnet in seiner eigenen Schrittweite (Mikroschritte), und nur zu den Makroschritten werden die Signale synchronisiert. Normalerweise werden bei dieser Synchronisation auch die Simulationsdaten der Mikroschritte mit übertragen, was natürlich länger dauerte. In MC kann man nur die Übertragung der Makrodaten aktivieren. Da die Simulationsergebnisse nur in AVLab ausgewertet wurden, konnte man für jeden Port in MC das Aufzeichnen der Ergebnisse deaktivieren, sowie keinen *Debug Log* oder keine Ergebnisdatei erstellen. Jeder dieser Schritte verbesserte die Gesamtlaufzeit der Simulation, sodass sie schlussendlich schneller als Echtzeit war.

6.9 Herausforderungen

Um Probleme mit der Laufzeit so gut als möglich zu vermeiden, wurden die Simulationen auf einer Workstation, durchgeführt. Dieser hatte 32 Kerne mit je 2,6GHz und 64 GB RAM Speicher.

Die Ausführungszeit des Plant Modells war auf diesem PC nicht viel schneller, da alle Module nur sequenziell ausgeführt und somit nur auf einem Kern gerechnet werden konnten. Im Plant mussten nicht nur die Fahrzeugreaktionen berechnet werden, es wurde auch für jede *ICOS* Schnittstelle eine S-Funktion ausgeführt. Um die Rechenzeit auf mehrere Kerne zu verteilen, wurden die Signale aus dem *SwSys_Test* zu einem Vektor zusammengefasst und über zwei *ICOS* Blöcke geschrieben bzw. gelesen. In einem weiteren Simulink Modell wurde der Vektor eingelesen und in seine Einzelsignale aufgespalten. Dieses Modell wurde nach dem Plant in MC eingebaut und hatte den Vorteil, dass nun ein Kern das Plant berechnen konnte und ein weiterer die vielen *ICOS* Schnittstellen bediente (Abbildung 6.20).

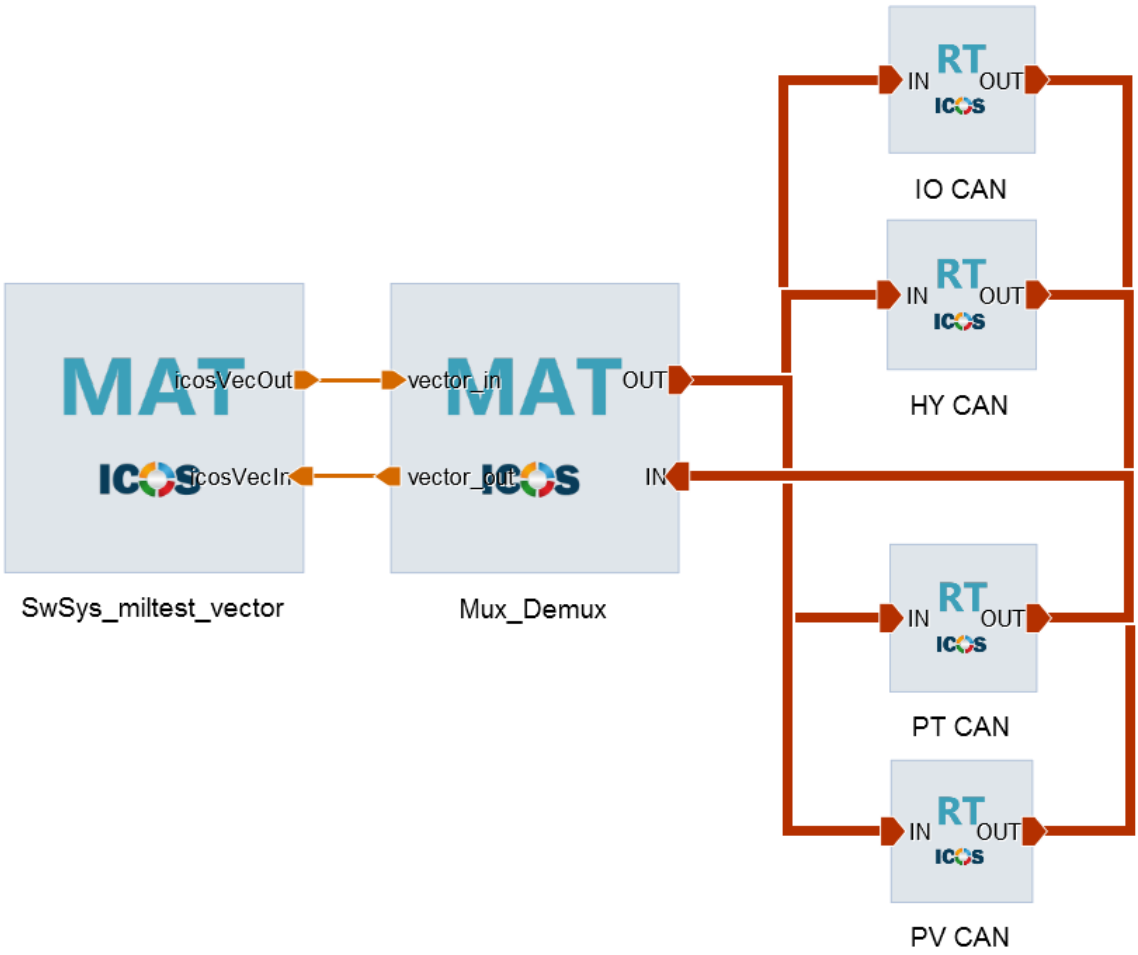

Abbildung 6.20: Aufbau in MC mit zusätzlichem Modell für Vektorextraktion zu einzelnen Signalen

Für die Laufzeittests wurden auf alle Eingänge des *SwSys_Tests* in MC Konstanten geschrieben und die Ausgänge offen gelassen. Auf der Workstation ergab sich für diese Anwendung ein Echtzeitfaktor von 0,31. Der Test einer Partnerfirma mit einem noch schnelleren Computer, dieser hatte 4GHz, führte sogar zu einem Faktor von 0,21.

Um diese Zeiten zu erreichen, wurden in MC für die Anwendung unnötige Funktionen deaktiviert. Die Deaktivierung der Signalverlaufspeicherung in MC stellte kein Problem dar, da die Ergebnisse mit AVLab ausgewertet wurden.

7 Zusammenfassung und Ausblick

Mit dieser Masterarbeit ist ein praktischer Anwendungsfall entstanden, den es in dieser Art noch nicht gegeben hat: das relativ junge Co-Simulationsprogramm Model.CONNECT wurde laufend weiterentwickelt sowie verbessert, auch aufgrund von Fehlerberichten und Wünschen, welche im Laufe der Arbeit aufgetreten sind. So wurde bisher immer nur eine CAN Verbindung mit ein paar wenigen Signalen verwendet, was natürlich ungleich einfacher und weit weniger komplex im Gegensatz zu den entstandenen Modellen war. Auch die zusätzliche Verwendung von AVLab in diesem Zusammenhang ist neu und die Kompatibilität der Systeme musste erst erforscht werden.

Die Änderungen in den ursprünglichen Simulink Modellen, um sie co-simulations-tauglich zu machen, ist überschaubar. Der größte Aufwand wurde für die Erstellung des Konzepts betrieben und für den ersten manuellen Aufbau eines Testsystems. Es ist einfach möglich, die Schnittstellen in *SwSys_Test* sowie den „Rahmen“ automatisch zu generieren und somit innerhalb kürzester Zeit eine neue Testumgebung für weitere Projekte zu schaffen. Voraussetzung dafür sind die Existenz einer DBC Datei und einer Excel Datei, in welcher die Zuordnung zwischen Simulink Signalnamen und CAN Signalnamen gegeben ist.

Für viele Signale und Parameter im Steuergerät existiert ein überschreibbarer Wert, um sie dauerhaft auf einen fixen Wert zu setzen. So kann man den Spannungslevel der konventionellen Autobatterie kontinuierlich auf einen plausiblen Wert setzen.

Zwischen dem IBS und der HVCU wurde über einen LIN Bus kommuniziert. Für die Simulation wurden die Signale im Fahrzeugmodell erzeugt und sollten an die LIN Schnittstelle des Steuergeräts gesendet werden. Die Idee war, die IBS Signale in MC über CAN an einen CAN zu LIN Konverter, der ebenfalls von PEAK Systems stammte, zu schicken, da man in MC einen CAN Bus einfach einbinden kann. Durch die geringe Priorität des Sensors in der Simulation wurden vorerst dessen geplante Signale in der HVCU als fixe Parameter überschrieben.

Im Zusammenspiel zwischen Steuergerät und Fahrzeugmodell treten Regelschleifen auf, etwa die Stromregelung der Hydraulikventile. Die erhöhten Laufzeiten der Signale bringen in diese Regelsysteme unerwünschte Totzeiten ein. Um die Auswirkungen dieser Zeiten zu verringern ist es bei den Signalen in MC möglich, intelligente Extrapolationsmechanismen zu aktivieren. Für den ersten Versuch wurden alle Signale

mit der einfachsten Möglichkeit, ZOH, extrapoliert. Für eine Verbesserung der Simulationsergebnisse wird es notwendig sein, jene an Regelkreisen beteiligten Signale zu finden und speziell zu behandeln

Bei der Anpassung der Simulink Modelle an die neue Testumgebung wurden Funktionalitäten hinzugefügt und entfernt. Es ist zu prüfen, ob im *SwSys_Test* Modell die *TargetLink* Ebene noch notwendig ist. Ursprünglich befand sich darunter die Steuergerätesoftware als referenziertes Modell, aus welcher der Programmcode für die HCU generiert wurde. Durch die Auftrennung der Modelle wird mit diesem Testmodell keine Steuergerätsoftware mehr erzeugt. Versuche haben gezeigt, dass hier noch Laufzeitverbesserungen möglich sind.

Mit Ende der vorliegenden Arbeit ist es möglich, mehrere hundert CAN Signale über MC zu senden und trotzdem ein echtzeitfähiges Simulationsmodell zu erhalten. Auch der Einsatz von mehreren CAN Bussen wurde implementiert. Das Hindernis für eine gelungene Simulation lag in der CAN Kommunikation. Bei der verwendeten Byte-Ordnung Motorola wurden von MC nicht die Bytes vertauscht, sondern die Signale innerhalb einer Nachricht. Durch diese Fehlfunktion empfing das Steuergerät keine sinnvollen Signale und wurde nicht gestartet. Auch das Auftreten hoher Buslasten durch die Gleichsetzung der Makroschrittweite von RT Blöcken mit Zykluszeiten von CAN Nachrichten war ein kritischer Punkt, welcher derzeit noch nicht gelöst ist.

Anhand dieses Projekts wurde gezeigt, dass ein einfacher Test eines Steuergeräts bereits in einem frühen Stadium der Entwicklung möglich ist, ohne dafür wertvolle Vorbereitungs- und Arbeitszeit in den HiL-Laboren aufzuwenden. Die Erweiterung des Aufbaus um mehrere Steuergeräte ist mit Anpassungen des Fahrzeugmodells möglich. Damit ist eine annähernd realistische Simulation des Fahrzeugverhaltens mit einfachen Mitteln und geringem zusätzlichen Programmieraufwand umsetzbar.

Abbildungsverzeichnis

Literaturverzeichnis

[1] H.-H. Braess und U. Seiffert, „Vieweg Handbuch Kraftfahrzeugtechnik“, Buch, Vieweg+Teubner, Auflage: 6, 2012, ISBN: 978-3-8348-8298-1.

[2] H. Richter, „Elektronik und Datenkommunikation im Automobil“, Technischer Bericht, lfl Technical Report Series, Institut für Informatik, lfI-09-05, Technische Universität Clausthal, 2005.

[3] M. Benedikt, J. Zehetner, D. Watzenig und J. Bernasch, „Moderne Kopplungsmethoden - Ist Co-Simulation beherrschbar?“, Online Magazin*, NAFEMS,* Nr. 22, pp. 63-74, Juli 2012.

[4] G. Weiß, „Zukünftige Softwarearchitekturen für Fahrzeuge“, Fraunhofer Institut für Eingebettete Systeme und Kommunikationstechnik, November 2012. [Online] http://www.esk.fraunhofer.de/content/dam/esk/dokumente/PDB_adaptives_Bordnetz_dt_web.pdf. [Zugriff am 21.Juli 2016].

[5] M. Benedikt, D. Watzenig und J. Zehetner, „Steuergeräte-Funktionsentwicklung durch Co-Simulation und Modellbibliothek“, Zeitschrift*, Automobiltechnische Zeitschrift,* Bd. 03, Nr. 10, 2015.

[6] W. D. Pietruszka, „MATLAB® und Simulink® in der Ingenieurpraxis“, Buch, Springer Vieweg, Auflage: 4, 2014, ISBN: 978-3-658-06420-4.

[7] D. Clemens, „Parallele Echtzeitsimulation mechatronischer Systeme“, Forschungsbericht, Gerhard-Mercator-Universität - GH Duisburg, Duisburg, 1993.

[8] M. Geimer, T. Krüger und P. Linsel, „Co-Simulation, gekoppelte Simulation oder Simulatorkopplung?“, Zeitschrift*, O + P Zeitschrift für Fluidtechnik,* Band 50(11-12), pp. 572-576, 2006.

[9] R. Schmoll, „Co-Simulation und Solverkopplung“, Dissertation, Fachbereich Maschinenbau der Universität Kassel, Kassel, 2015.

[10] M. Benedikt, D. Watzenig, J. Zehetner und A. Hofer, „NEPCE - A nearly energy-preserving coupling element for weak-coupled problems an co-simulation“, Konferenzbeitrag, in *International Conference on Computational Methods for Coupled Problems in Science and Engineering*, Ibiza, 2013.

[11] P. Hofmann, „Hybridfahrzeuge“, Buch, Springer, Auflage: 2, 2014, ISBN: 978-3-7091-1780-4.

[12] M. Arnold, „Multi-Rate Time Integration for Large Scale Multibody System Models“, Buch, Springer, Auflage: 1, 2007, ISBN: 978-1-4020-5981-0, pp. 1-10.

[13] F. González, M. Ángel Naya, A. Luaces und M. González, „On the effect of multirate co-simulation techniques in the efficiency and accuracy of multibody system dynamics“, Buch, Springer, Auflage: 1, 2011, ISSN: 1573-272X, pp. 461-483.

[14] S. S. Sadjina und E. Pedersen, „Energy Conservation and Coupling Error Reduction in Non-Iterative Co-Simulations“, Publikation in Begutachtung, Trondheim, 2016.

[15] J. Zehetner, G. Stettinger, H. Kokal und B. Toye, „Echtzeit-Co-Simulation für die Regelung eines Motorprüfstands“, Zeitschrift*, Automobiltechnische Zeitschrift,* Band 116, 2014.

[16] N. Thek, M. Benedikt und J. Zehetner, „Co-Simulation under Hard-Real-Time Conditions“, Konferenzbeitrag, in *NAFEMS World Congress 2013*, Salzburg, 2013.

[17] K. Reif, „Automobilelektronik“, Buch, Springer Vieweg, Auflage: 5, 2014, ISBN: 978-3-658-05048-1, pp. 14-19.

[18] M. Zauner und A. Schrempf, „Informatik in der Medizintechnik“, Buch, Springer-Verlag, Auflage:: 1, 2009, ISBN: 978-3-211-89189-6.

[19] The MathWorks, 11.Juli 2016. [Online] http://de.mathworks.com/products/matlab/

[20] The MathWorks, 11.Juli 2016. [Online] http://de.mathworks.com/products/simulink/

[21] D. Louarn-Pioch, „*Function developement and test environment: AVLab, integrating Visu-IT ADD*“, Internes Dokument, nicht veröffentlicht, Deutschland, 2014.

[22] AVL List GmbH, „*Model.CONNECT™ User Manual*“, Handbuch, 2016.

[23] The MathWorks, 11.Juli 2016. [Online] http://de.mathworks.com/help/simulink/ug/virtual-and-nonvirtual-buses.html

[24] J. Weber, „*P2 Full Hybrid E/E System Specification*“, Internes Dokument, nicht veröffentlicht, AVL List GmbH, 2015.

[25] L. Stiny, „Aktive elektronische Bauelemente“, Buch, Springer, Auflage: 2, 2015, ISBN: 978-3-658-09153-8, pp. 221-224.

[26] Texas Instruments, „*LM6132/LM6134 Dual and Quad Low Power 10 MHz Rail-to-Rail I/O Operational Amplifiers*“, Datenblatt, Texas, 2000.