Entwurfsdokument

Service-Interface für ein Formula-Student-Fahrzeug

Technische Universität Ilmenau
Softwareprojekt SS 2013
Gruppe 19

Christian Boxdörfer
Thomas Golda
Daniel Häger
David Kudlek
Tom Porzig
Tino Tausch
Tobias Zehner
Sebastian Zehnter

Hier Datum einfügen

betreut durch

Dr. Heinz-Dietrich Wuttke, TU Ilmenau
Oliver Dittrich, fachlicher Betreuer Team StarCraft e.V.

Inhaltsverzeichnis

1 Einleitung

Hier Text einfügen!

2 Randbedingungen

Hier Text einfügen!

3 Grobentwurf

Hier Text einfügen!

3.1 Architekturmuster und Systemzerlegung

3.1.1 dSPACE MicroAutoBox II

3.1.2 Embedded-PC

3.1.3 Webserver

3.1.4 Webseite

3.1.5 Datenbanken

Fahrzeugdaten-Datenbank

Benutzerdaten-Datenbank

3.2 Simulink-Modell

3.2.1 Verwendete Blöcke

Um den Einstieg in unsere im folgenden aufgeführte Modellausschnitte zu erleichtern, werden im Verlauf dieses Abschnittes alle zur Erstellung des Simulink-Modells für die dSPACE MicroAutoBox II verwendeten Blöcke vorgestellt und ihre Funktionsweise kurz erläutert.

Sources

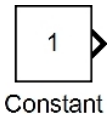

Abbildung 3.1: Constant-Block

Abbildung 3.2: Repeating Sequence Interpolated - Block

1) *Constant-Block*

   Der Constant-Block ermöglicht die Generierung eines reellen oder komplexen konstanten Wertes. Je nach Modifikation der Einstellungen des Blocks wird es zudem ermöglicht, neben einem konstanten Skalar einen konstanten Vektor oder eine konstante Matrix als Eingangssignal bereitzustellen. Als Datentypen für das Eingangssignal stehen die unter X.Y. aufgeführten Datentypen zur Verfügung.

2) *„Repeating-Sequence-Interpolated" - Block*

Im Gegensatz zum Costant-Block ermöglicht dieser Block die Erzeugung eines individuellen, sich periodisch wiederholenden und kontinuierlichen Signals mittels einer Interpolation anhand zuvor selbst definierter diskreter Zeit- und Funktionswerte, welche in zwei Vektoren gleicher Länge gespeichert sind.

Ports & Subsystems

Abbildung 3.3: Inport-Block

Abbildung 3.4: Outport-Block

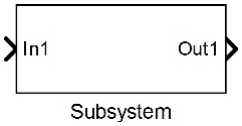

Abbildung 3.5: Subsystem

1) *Inport-Block*

Dieser Block hat in unserem Modell die Aufgabe, die zuvor festgelegten Eingangssignale des Subsystems auf der Modellebene des Subsystems selbst zu repräsentieren. Darüber hinaus mit ist es in der Top-Level-Ebene des Systems mit diesem Block auch möglich, externe Eingangssignale aus dem Arbeitsbereich für das Modell bereitzustellen.

2) *Outport-Block*

Die Aufgabe des Outport-Blockes ist es, eine Verknüpfung vom aktuellen System zu einem Zielsystem außerhalb der Modellebene herzustellen.

3) *Subsystem*

Innerhalb eines Subsystems können verschiedene Blöcke zusammengefasst werden, was eine Strukturierung und Gliederung der Signalflüsse erleichtert und zudem eine deutlich übersichtlichere Darstellung des Modells zur Folge hat. Weiterhin ist es auch möglich, mehrere Subsysteme in einem Subsystem zusammenzufassen, um eine beliebige Tiefe innerhalb der Hierarchie eines Modells zu realisieren.

Signal Routing

1) *Bus Creator*

Mit Hilfe eines solchen Bus Creators wird es ermöglicht, mehrere Signale (a, b, c) zu einem Gesamtsignal (d_gesamt) zu bündeln.

2) *Bus Selector*

Umgekehrt erlaubt es der Bus Selector, aus einem Gesamtsignal wieder einzelne Signale zu selektieren und diese gesondert weiterzuleiten. So wird wie in Abb. X.Y. dargestellt wieder aus dem Gesamtsignal d_gesamt die Signale a, b und c herausgeführt.

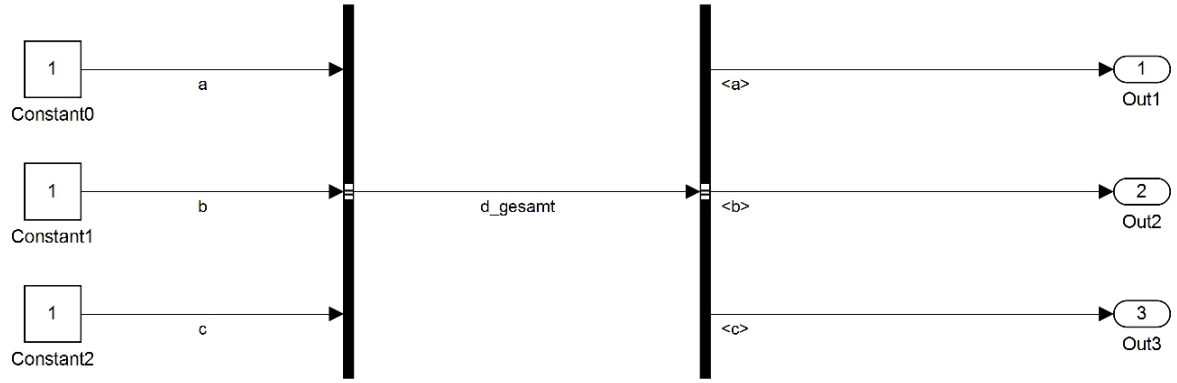

Abbildung 3.6: Bussystem bestehend aus einem Bus Creator (l.) und einem Bus Selector (r.)

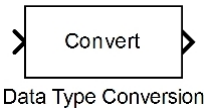

Abbildung 3.7: Convert-Block

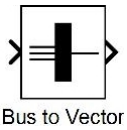

Abbildung 3.8: Bus to Vector - Block

Signal Attributes

1) *Convert-Block*

   Mit dem Convert-Block können verschiedene Anforderungen realisiert werden:

   – Konvertierung eines Signals bzw. eines Signalvektors in einen anderen Datentyp, hierbei kann die Art der Rundung selbst festgelegt werden.

   – Umbenennung eines Signals bzw. eines Signalvektors.
   Dies hat den Zweck, dass man nach dem Convert-Block unabhängig vom angelegten Eingangssignal mit einem festen Datentyp und einem fest vergebenem Variablennamen in anderen Systemen bzw. auf anderen Plattformen arbeiten kann.

2) *„Bus to Vector" - Block*

   Dieser Block konvertiert ein virtuelles Bussignal in ein Vektorsignal. Hierbei ist es erforderlich, dass alle am zu konvertierenden Bus anliegenden Signale im Datentyp, Signaltyp und im gewählten Sampling-Verfahren übereinstimmen.

Sinks

Abbildung 3.9: Terminator-Block

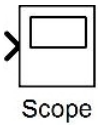

Abbildung 3.10: Scope

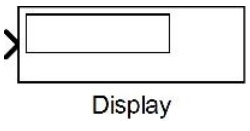

Abbildung 3.11: Display

1) *Terminator*

   Der Terminator dient dazu, die Signale, deren Ausgänge nicht mit Blöcken o.ä. verbunden sind, abzuschließen.

2) *Scope*

   Der Scope-Block stellt den Signalverlauf eines Signals über der Simulationszeit dar, weswegen er sich hervorragend dafür eignet ein erstelltes Modell auf dessen Korrektheit zu überprüfen.

3) *Display*

   Der Display-Block zeigt den konkreten Wert eines Eingangssignals an. Um die Anzeige des Displays anzupassen, kann der Entwickler über die Einstellungen der Format-Parameter Einfluss darauf nehmen.

Math Operations

1) *Gain-Block*

   Der Gain-Block multipliziert das Eingangssignal mit einer selbst gewählten Konstante, wobei das Eingangssignal selbst ein Skalar, ein Vektor oder eine Matrix sein kann.

dSPACE-Blöcke

Im Folgenden sollen nun diejenigen Blöcke vorgestellt werden, welche speziell für eine Verwendung mit der MicroAutoBox II konzipiert wurden und u.a. für eine Kommunikation ebendieser mit dem Embedded-PC unerlässlich sind.

1) *Encode32-Block*

2) *„UDP Send" - Block*

3.2.2 Signalgenerator

Das Modell, welches auf der dSPACE MicroAutoBox II implementiert wird, besitzt zwei Subsysteme - den Signalgenerator und den Signalkollektor. Der Signalgenerator hat hierbei die Aufgabe, auf der MicroAutoBox II die für einen Systemtest benötigten Testsignale zu generieren, welche anschließend zum Embedded-PC via UDP-Schnittstelle gelangen und daraufhin per GPRS / UMTS an einen vServer gesendet werden. Dieses Subsystem wird jedoch nach einem erfolgreichen Test des Service-Interfaces durch ein Subsystem von Team Starcraft e.V. ersetzt, welches im Stande ist die im Formula-Student-Fahrzeug verbauten Komponenten anzusprechen und somit im Unterschied zu dem aktuell verwendeten Signalgenerator statt künstlich erzeugter Daten die realen Daten auszulesen und an den Signalkollektor weiterzuleiten.

Struktur des Subsystems

Um ein hohes Maß an Übersichtlichkeit und Modularität zu gewährleisten, wird an den Vorgaben von Team StarCraft e.V. orientierend das Subsystem des Signalgenerators nochmals in einzelne Subsysteme unterteilt, die die verschieden Kategorien der jeweiligen Fahrzeugdaten repräsentieren.

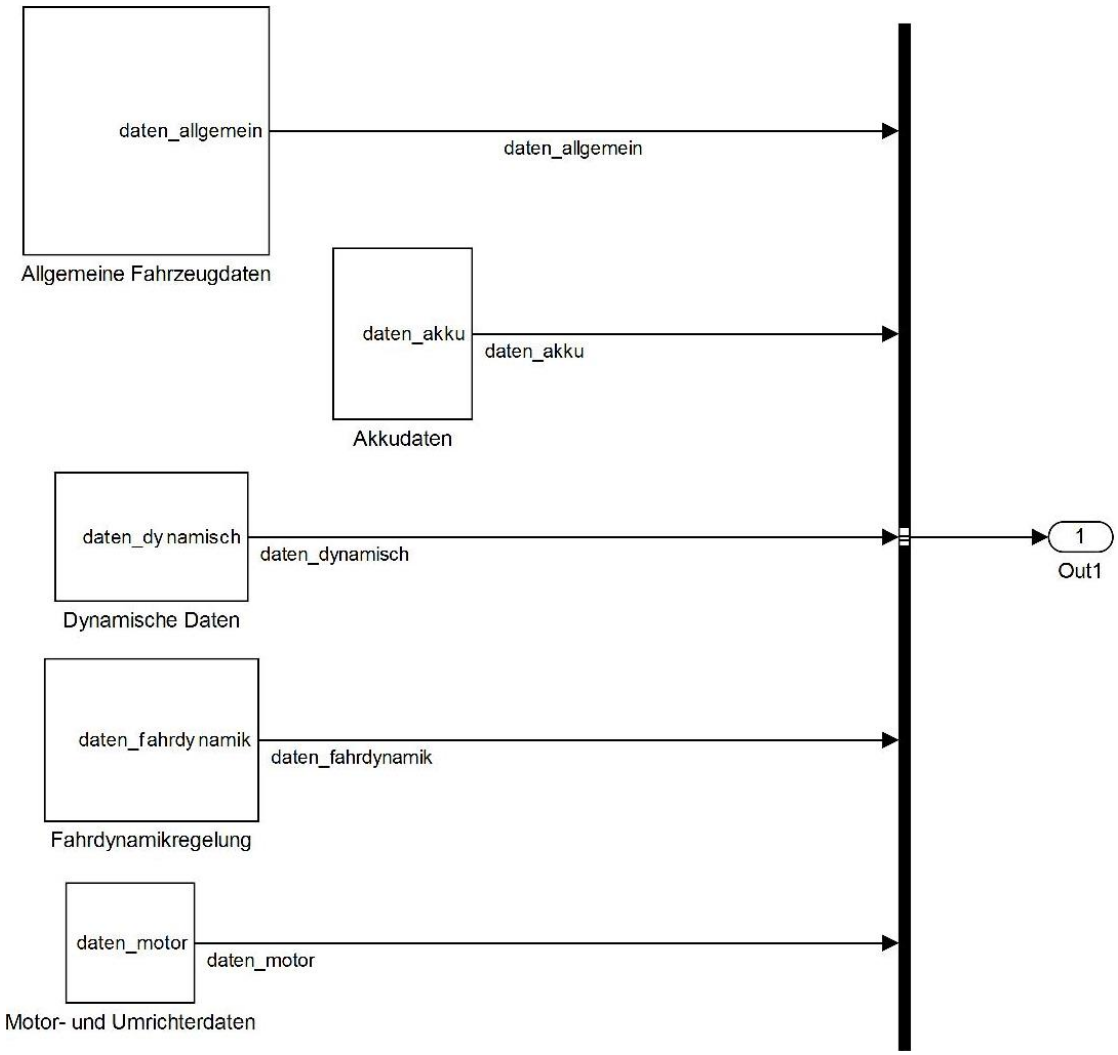

Abbildung 3.12: Subsysteme des Signalgenerators nach den Kategorien der Fahrzeugdaten

Innerhalb dieser Subsysteme findet nun - ohne Beschränkung der Allgemeinheit am Subsystem „Daten Allgemein" erläutert - die Signalerzeugung (Geschwindigkeit, Gesamtspannung des Akkus und der Fahrzeugzeit) statt, wobei wie in X.Y dargestellt die Signalerzeugung mehrere Signale der gleichen „Signalgruppe" wiederum zu Subsystemen (Notausfunktionen, Temperaturen und Gaswerte) innerhalb des Subsystems „Allgemeine Daten" zusammengefasst werden. Die eigentliche Erzeugung der Signale wird wie in Abbildung X.Y. gezeigt mittels Blöcken aus der Kategorie „Sources"(s. X.Y) realisiert. Hierbei wird je nach Input entweder ein Constant-Block mit dem Datentyp *boolean* zur Realisierung von Schaltern etc. (s. Notausfunktionen)oder ein Repeating Sequence Interpolated (RSI) - Block mit dem Datentyp *single* zur Modellierung der restlichen Fahrzeugkomponenten verwendet (s.

Gaswerte). Der Datentyp *single* wurde deshalb gewählt, um auch wie von Team StarCraft e.V. gefordert als Input Werte mit mehreren Nachkommastellen realisieren zu können. Die auf analoge Weise zu X.Y mit Constant- oder RSI-Blöcken erzeugten 386 Input-Signale werden schließlich im Umkehrschluss zur obigen Beschreibung über mehrere Bussysteme und Subsysteme zu einem Busarray im Signalgenerator zusammengefasst, welcher nunmehr dieses an den Signalkollektor übergibt.

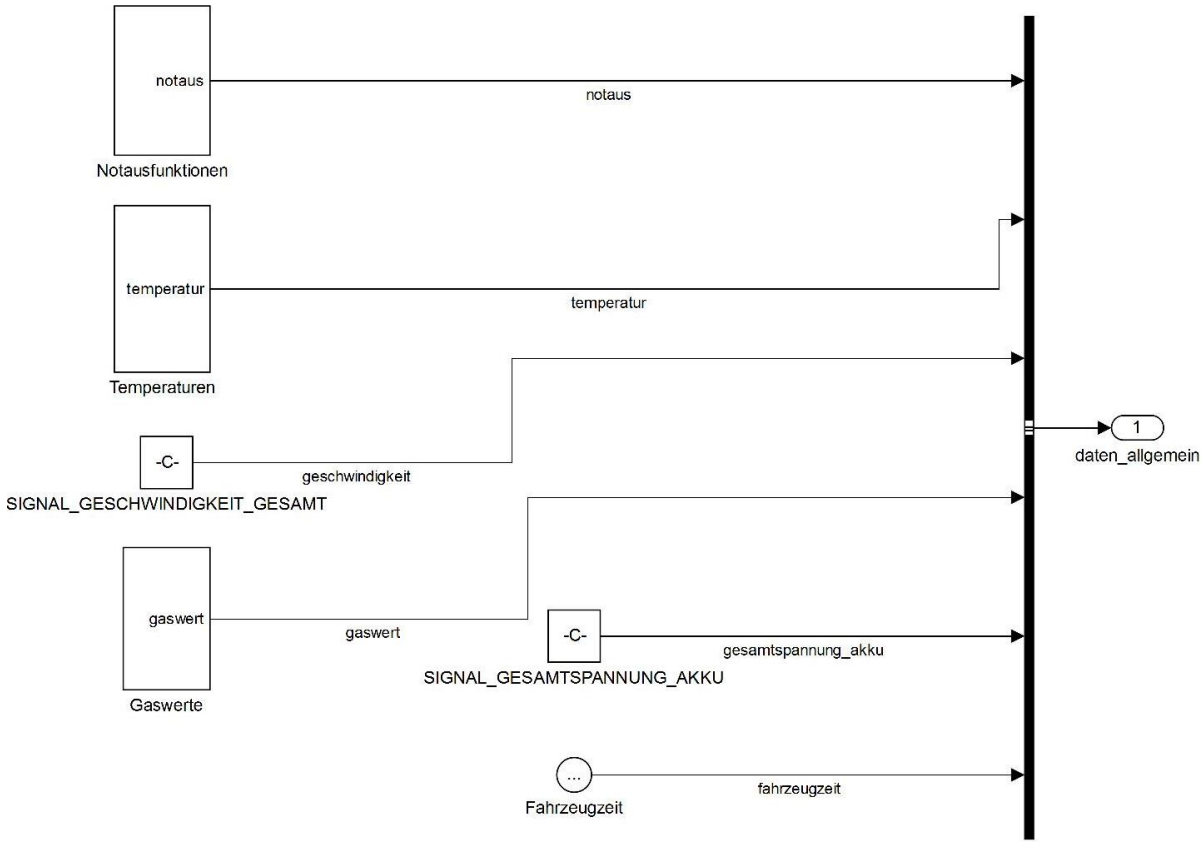

Abbildung 3.13: Subsystem „Allgemeine Daten" im Signalgenerator

Config-Datei „signalgenerator_microautobox.m"

Um eine mögliche Änderung der Testsignale zu vereinfachen und eine übersichtliche Darstellung aller Testsignale zu realisieren, sorgt eine Konfigurationsdatei in Matlab für die Spezifizierung der Testsignale des Simulink-Modells. Die in dem *.m-File aufgeführten Parameter für die Zeit- und Funktionsvektoren dienen als Referenz für die signalerzeugenden Blöcke, welche über den Workspace von Matlab auf diese nach dem Ausführen der Datei zugreifen können (s. Abb. X.Y). Hierbei sollte jedoch beachtet werden, dass vor dem Compilieren des Simulink-Modells und der Implementierung ebendiesem auf der MicroAutoBox II einmalig das *.m-File ausgeführt werden muss. Demzufolge muss auch nach dem Ändern von Parametern die Datei neu ausgeführt werden, damit bei einer erneuten Implementierung des Simulink-Modells die geänderten Parameter korrekt übernommen werden können. Zudem wurde auch um die Benennung der Parameter im *.m-File zu standardisieren jeder Funktionsvektor mit dem Präfix „SIGNAL_" und jeder Zeitvektor mit dem Präfix „TIME_" versehen.

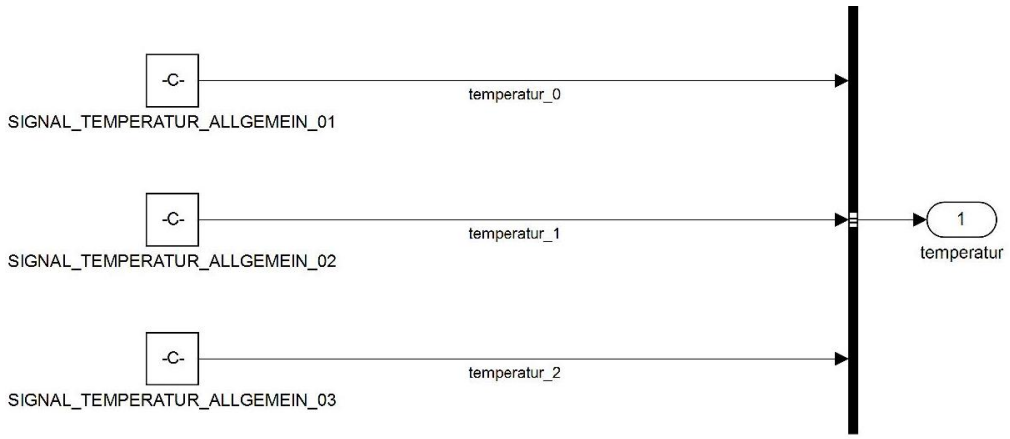

Abbildung 3.14: Signalerzeugung im Subsystem "Temperaturen"

3.2.3 Signalkollektor

Der Signalkollektor stellt wie bereits erwähnt das zweite große Subsystem innerhalb des Simulink-Modells dar. Er hat die Aufgabe den vom Subsystem „Signalgenerator" (s. X.Y) oder von einem späteren Subsystem von Team StarCraft e.V. erhaltenen Busarray wieder in die einzelnen Signale zu unterteilen, ggf. geeignet aufzubereiten und diese dann eine UDP-Schnittstelle innerhalb des Subsystems weiterzuleiten, welche die erzeugten Testsignale bzw. Testdaten an den Embedded-PC sendet.

Struktur des Subsystems

Das Subsystem selbst besitzt die folgende innere Struktur:
In einem ersten Schritt wird das Busarray wieder in den Kategorien A) - E) der Fahrzeugdaten entsprechenden Signalgruppen aufgeteilt und den fünf Subsystemen (vgl. Abb. X.Y) zugeführt. Dort werden die nunmehr fünf Busarrays wieder über mehrere Busselektoren und Subsysteme hinweg in die 368 einzelnen Signale aufgelöst, welche somit einzeln aufbereitet werden können (s. Abb. X.Y). Während der Aufbereitung der Signale werden die folgenden Schritte durchgeführt:

1) *Verstärkung der Signale*

   Abhängig von der Anzahl der Nachkommastellen des Testsignals n *mit* $n > 0$ wird dieses nun durch einen Gain-Block (s. X.Y) mit dem Faktor 10^n multipliziert, um für den aktuellen Wert des Testsignals jeweils einen ganzzahligen Wert zu erhalten, was zur Vereinheitlichung der zu übertragenden Daten beiträgt.

2) *Konvertierung der Datentypen*

   Die nunmehr ganzzahligen Werte werden von ihren Datentypen *boolean* oder *single* nun einheitlich mittels eines Convert-Blocks (s. X.Y) in den Datentyp *int16* konvertiert, wodurch somit alle Signale den gleichen Datentyp aufweisen.

3) *Umbenennung der Variablennamen*

   Mit dem Convert-Block ist es zudem möglich, dem Ausgangssignal unabhängig vom Eingangssignal einen festen bzw. neuen Variablennamen zu vergeben. Dies ist insofern

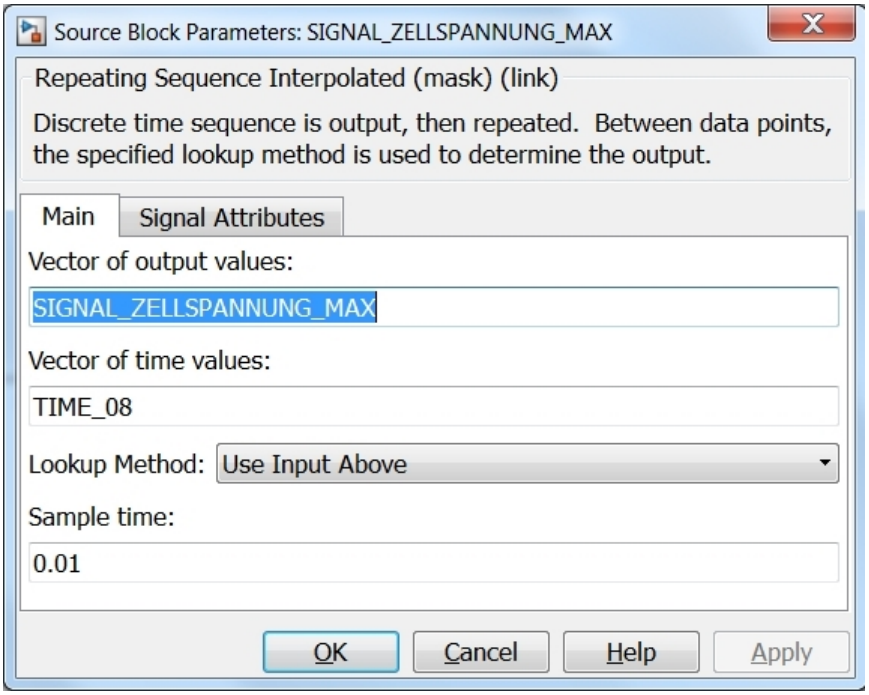

Abbildung 3.15: Referenzierung im RSI-Block

nützlich, da bei einem möglichen Zugriff auf die Daten bzw. die Variablen durch den Embedded-PC diese immer die gleichen Variablennamen besitzen, unabhängig davon ob der Signalgenerator durch das Subsystem von Team StarCraft e.V. ersetzt wurde oder nicht.

Nachdem die Daten anhand der obigen Schritte aufbereitet wurden, werden dieser erneut durch mehrere Bus Creator und Subsysteme auf einem zentralen Bus Creator zu einem Busarray gebündelt und auf einen Bus to Vector - Block (s. X.Y.) gegeben, welcher das Busarray in einen Vektor umwandelt. Anschließend wird dieser Vektor der UDP-Schnittstelle zugeführt.

3.2.4 UDP-Schnittstelle

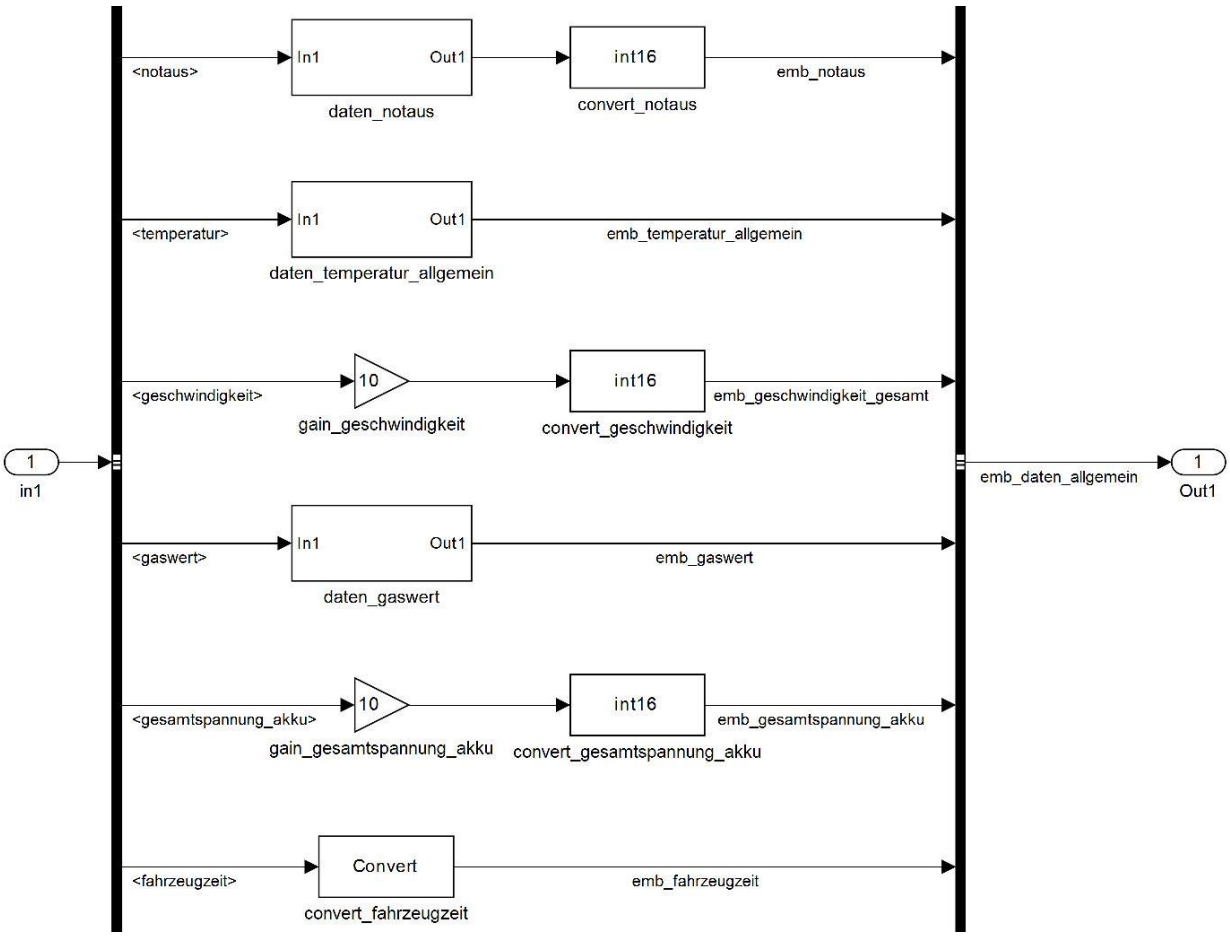

Abbildung 3.16: Signalaufbereitung im Subsystem „daten_allgemein"

4 Feinentwurf

Glossar

Abbildungsverzeichnis

Anhang